Mikko Homanen

Sovellus tietovarastokehityksen automatisointiin ja metadatan hallintaan

Tietotekniikan pro gradu -tutkielma

23. joulukuuta 2019

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Mikko Homanen

Yhteystiedot: mikko.a.homanen@student.jyu.fi

Ohjaaja: Ville Isomöttönen

Työn nimi: Sovellus tietovarastokehityksen automatisointiin ja metadatan hallintaan

Title in English: Application for data warehouse development automation and metadata management

Työ: Pro gradu -tutkielma

Opintosuunta: Ohjelmistotekniikka

Sivumäärä: 68+0

Tiivistelmä: Tässä tutkielmassa kuvataan suomalaisen IT-palveluita ja ohjelmistoratkaisuja tarjoavan yrityksen tarpeisiin toteutettu sovellus tietovarastokehityksen automatisointiin ja metadatan hallintaan. Sovelluksella yritys pyrki saamaan kilpailuetua sekä mahdollistamaan kustannustehokkaiden ja laadukkaiden tietovarastoprojektien toteutuksen. Tutkimus toteutettiin suunnittelutieteellisenä tutkimuksena, jonka tavoitteena oli tuottaa mahdollisimman hyvin yrityksen tavoitteita vastaava sovellus. Sovellusta arvioimalla tuodaan esille, kuinka hyvin se vastaa sille asetettuja kriteereitä. Tutkielmassa esitetään myös erilaisia keinoja automatisoida tietovarastokehitystä sekä tuodaan esille etuja, joita automatisoinnilla voidaan saavuttaa.

Avainsanat: tietovarastointi, tietovarasto, automatisointi, metadata, data vault, suunnittelutiede

Abstract: This thesis describes a data warehouse automation software developed for the needs of a Finnish IT services and software solutions company. With the application, the company pursued to gain competitive advantage and to enhance its ability to produce high quality and cost effective data warehousing projects. The thesis was carried out as a design science research with the purpose to produce an application that suits the needs of the company as well as possible. The applications ability to fulfill its purpose is evaluated against

criteria set before and during development. The thesis also describes different means of automating data warehouse development process and the benefits of utilizing automation in data warehouse in development.

Keywords: data warehousing, data warehouse, automation, metadata, data vault, design science

Kuviot

Taulukot

Sisältö

1 Johdanto

Yritykset ovat pyrkineet hyödyntämään dataa päätöksenteon tukena jo kymmeniä vuosia. Organisaatiot ovat 1990-luvulta lähtien keskittäneet päätöksenteon tukena käytettävää dataa *tietovarastoihin*, jotka ovat osa tiedolla johtamisen kokonaisuuksia, *Business Intelligence* -järjestelmiä (Vaisman ja Zimányi 2014). Vaikka tietovarastointi ja Business Intelligence ovat molemmat kehittyneet jo 1990-luvun puolella, on viimeisen vuosikymmenen aikana tiedolla johtamisen ja tiedon analysoinnin rooli kasvanut entistäkin olennaisempaan rooliin nykyaikaisissa yrityksissä. Esimerkiksi McAfeen ym. (2012) mukaan datan hyödyntämistä painottavat yritykset voivat olla jopa 5% tehokkaampia ja 6% tuottavampia kuin kilpailijansa. Taustatekijöinä datan hyödyntämisen tuomille eduille McAfee ym. mainitsevat esimerkiksi 2010-luvulle tultaessa kasvaneen datan määrän sekä määrän kasvunopeuden merkittävän kasvun. Datan määrän lisäksi datan rakenne on monipuolistunut tuoden samalla uusia mahdollisuuksia mahdollisuuksia datan hyödyntämiselle (McAfee ym. 2012; Agarwal ja Dhar 2014). Yritykset keräävät, analysoivat ja hyödyntävät tietoa useista eri liiketoimintaprosesseista sisältäen perinteisten talousdatan ja henkilöstödatan lisäksi nykyisin myös paljon esimerkiksi IoT-dataa sekä ei-strukturoitua dataa, kuten kuvia ja videoita (McAfee ym. 2012; Agarwal ja Dhar 2014).

Datan määrän kasvu ja rakenteen monipuolistuminen on vaikuttanut myös tietovarastointiin. Tietolähteiden ja tiedon rakenteiden monipuolisuus sekä jatkuvasti muuttuvat vaatimukset vaativat myös tietovaraston kykyä muuntautua vaatimuksiin (Chandra ja Gupta 2018). Ohjelmistotuotannosta tutut ketterät menetelmät ovatkin tulleet olennaiseksi osaksi myös tietovarastointiprojektien läpivientiä (Larson ja Chang 2016). Linstedt ja Olschimke (2016) kuvaavat erityisesti tietovarastointiin tarkoitetun Data Vault -menetelmän, jossa hyödynnetään useita ketterien ohjelmistokehitysprosessien parhaita käytäntöjä. Data Vault -menetelmässä on olennaisessa osassa tietovaraston kehitys inkrementaalisesti aloittaen pienistä osista jatkuvasti laajentamalla, ja siinä painotetaan templaatteihin pohjautuvaa automaation hyödyntämistä (Linstedt ja Olschimke 2016, s. 12,17).

Tietovarastokehitystä pidetään yleisesti työläänä ja monimutkaisena prosessina (Rahman ja Rutz 2015). Tietovarastokehitykseen kuuluvaa manuaalista työtä on kuitenkin mahdollista

automatisoida. Etenkin tietovaraston populointiin liittyvien datan latausprosessien toteutusta on pidetty työläänä ja virhealttiina vaiheena, jossa voidaan hyödyntää paljon automaatiota (esimerkiksi Tomingas, Kliimask ja Tammet 2015). Puonnin ym. (2016) mukaan automatisoimalla voidaan tehostaa kehitystä merkittävästi sekä minimoida inhimillisten virheiden riskiä. Lisäksi he huomauttavat automatisoinnin helpottavan ratkaisun ylläpitoa ja jatkokehitystä, kun esimerkiksi arkkitehtuuriratkaisut ja nimeämiskäytännöt ovat yhtenäiset koko ympäristössä.

Varsinaiset tietovarastolatauksien toteuttamiseen tarjottavat teknologiat eivät yleensä mahdollista automaatiota suoraan, vaan vaativat paljon manuaalista työtä (Puonti ym. 2016; Simitsis, Vassiliadis ja Sellis 2005). Kehityksen automatisointi pohjautuu usein tietovaraston metadatan hallintaan ja hyödyntämiseen (esimerkiksi Puonti ym. 2016; Pankov ym. 2014). Automatisoinnin lisäksi metadataa voidaan hyödyntää dokumentoimaan tietovarastoympäristöä (Pankov ym. 2014). Toisaalta tietovarastoympäristö voi olla hyvin laaja, joten erillinen työkalu on tarpeen, jotta metadataa voidaan hyödyntää tehokkaasti (Sen ja Sinha 2005). Näistä syistä kehityksen automatisointia ja metadatan hallintaa varten tarvitaan erityisesti siihen tarkoitettuja sovelluksia.

Tässä tutkielmassa kuvataan suomalaisen IT-palveluita ja ohjelmistoratkaisuja tarjoavan yrityksen tarpeisiin toteutettu sovellus tietovarastokehityksen automatisointiin ja metadatan hallintaan. Sovelluksella yritys pyrki saamaan kilpailuetua sekä mahdollistamaan kustannustehokkaiden ja laadukkaiden tietovarastoprojektien toteutuksen. Tutkimus toteutettiin suunnittelutieteellisenä tutkimuksena, jonka tavoitteena oli tuottaa mahdollisimman hyvin yrityksen tavoitteita vastaava sovellus. Sovellusta arvioimalla tuodaan esille, kuinka hyvin se vastaa sille asetettuja kriteereitä. Tutkielmassa esitetään myös erilaisia keinoja tietovarastoinnin automatisointiin sekä tuodaan esille etuja, joita automatisoinnilla voidaan saavuttaa.

Aluksi luvussa 2 esitellään yleisellä tasolla tietovarastointia ja siihen liittyviä käsitteitä, tiedon mallintamista, tietovarastoarkkitehtuureja sekä tietovarastojen populointia. Luvussa 3 tehdään kirjallisuuskatsaus aiempaan tutkimukseen tietovarastokehityksen automatisoinnista. Luvussa 4 esitellään suunnittelutieteellisen tutkimuksen menetelmiä ja niiden soveltamista tässä tutkielmassa. Luvussa 5 esitellään tarkemmin tutkimuksen tuloksena syntynyt automatisointisovellus. Luvussa 6 arvioidaan tutkimusprosessin ja tutkimusartefaktin onnis-

tumista. Lopuksi luvussa 7 tehdään yhteenveto tutkimuksesta ja sen tuloksista sekä esitetään mahdollisuuksia jatkotutkimukselle.

2 Tietovarastointi

Tässä luvussa esitellään ensin tutkielmassa käytettävä tietovaraston käsite sekä lyhyt katsaus tietovarastojen historialliseen taustaan. Seuraavaksi luvussa 2.2 esitellään tiedon mallinnusta yleisellä tasolla sekä erilaisten mallinnusmenetelmien käyttöä tietovarastoissa. Luvussa 2.3 tarkastellaan erilaisia tietovarastoarkkitehtuureja. Lopuksi luvussa 2.4 käsitellään tietovaraston populointia.

2.1 Taustaa ja määritelmä

Vaismanin ja Zimányin (2014, s. 4) mukaan tietovarastot syntyivät ratkaisemaan 1990-luvun alussa havaitun ongelman, kun organisaatioissa huomattiin laajalti yleistyvä tarve hienostuneelle datan analysoinnille päätöksenteon tueksi. Perinteiset operatiiviset tietokannat (OLTP, Online transactional processing) eivät Vaismanin ja Zimányin mukaan soveltuneet tällaiseen analysointiin, sillä niille on tyypillistä esimerkiksi huono suorituskyky, kun tietokantakyselyt kohdistuvat laajaan joukkoon tauluja tai ryhmittelevät suuria määriä dataa. Toisena olennaisena taustatekijänä he toivat esille tarpeen yhdistää usean eri operatiivisen järjestelmän dataa, kun organisaatiota halutaan analysoida kokonaisvaltaisesti.

Yleisesti käytetty määritelmä tietovarastolle on alunperin esitetty Bill Inmonin teoksessa *Building the data warehouse* (2002, s. 31-34), jonka ensimmäinen versio julkaistiin vuonna 1990. Inmonin määritelmän mukaan "tietovarasto on aihesuuntautunut, integroitu, aikariippuvainen ja muuttumaton kokoelma dataa, joka tukee liikkeenjohdon päätöksentekoa". *Aihesuuntautuneisuus* tarkoittaa, että tietovarasto käsittää johonkin aihepiiriin kuuluvia käsitteitä. Esimerkiksi vakuutusyhtiön tietovarasto voi sisältää käsitteitä kuten asiakas tai korvausvaatimus. Nämä eri aihepiirien datat voivat koostua useiden eri järjestelmien datasta, jolloin tietovaraston tulee olla *integroitu*. Integroidussa tietovarastossa eri järjestelmistä peräisin oleva, kenties alunperin eri muodossa käsitelty data yhdistetään yhden esitysmuodon alle. *Muuttumattomuudella* tarkoitetaan tässä yhteydessä sitä, että kerran tietovarastoon ladattua tietuetta ei päivitetä jälkikäteen. Operatiiviset järjestelmät päivittävät usein samaa tietuetta, kun taas tietovarastossa halutaan säilyttää historiatieto. Tämän vuoksi tietueiden muuttumisen yhtey-

dessä niistä kirjoitetaan tietovarastoon uusi versio muokkaamatta vanhaa. Historiatiedon tallennuksesta seuraa viimeinen kriteeri, *aikariippuvaisuus*. Aikariippuvaisuudella tarkoitetaan sitä, että jokainen tietovarastoon säilötty tietue esittää tilanteen tiettynä ajanhetkenä. Tämä mahdollistaa tiedon vertailun eri ajanhetkien välillä. Tällaisesta ratkaisusta käytetään joskus myös nimitystä *Enterprise Data Warehouse* (EDW)

Kemper ja Neumann (2011) kuvaavat operatiivisten järjestelmien tietokantoja transaktioiden, kuten pankkisiirtojen tai myyntimerkintöjen käsittelyyn tarkoitettuina järjestelminä. Edelleen he toteavat tällaisten transaktioiden kohdistuvan kerralla vain pieneen osaan datasta, jolloin niiden suoritus on nopeaa. Tällaisia operaatioita kutsutaan OLTP (Online Transaction Processing) -operaatioiksi, joita varten operatiiviset tietokannat on suunniteltu (Plattner 2009). Business Intelligence (BI, suom. joskus tiedolla johtaminen) -työkalut puolestaan suorittavat useimmiten kyselyjä, joissa huomattava osa datasta prosessoidaan kerralla liiketoimintaa analysoivien henkilöiden tarpeisiin (Kemper ja Neumann 2011). Tällaisia operaatioita kutsutaan OLAP (Online Analytical Processing) -operaatioiksi.

OLAP-operaatioiden suorittaminen suoraan operatiivisen järjestelmän tietokantaan voi aiheuttaa helposti resurssiongelmia, kun järjestelmän omat OLTP-operaatiot kilpailevat samoista resursseista (Kemper ja Neumann 2011). Resurssiongelmien seurauksena syntyi Kemperin ja Neumannin mukaan arkkitehtuuri, jossa tiedon analysointi OLAP-operaatioilla suoritetaan eri ympäristössä kuin operatiiviset järjestelmät. Data ladataan ETL (Extract, Transform, Load) -prosesseilla (ks. luku 2.4) perinteisesti määräajoin erilliselle tietovarastoalustalle (Castellanos ym. 2009). Latausprosessit on Castellanoksen ym. mukaan tyypillisesti suoritettu öisin, jolloin minimoidaan haitat lähdejärjestelmän kuormituksesta. Näin toimien BI-työkalujen suorittamat OLAP-operaatiot eivät estä varsinaisten lähdejärjestelmien käyttöä, mutta toisaalta data ei ole aina ajan tasalla, vaan käytettävissä on esimerkiksi aina data sellaisena kuin se oli raporttia edeltävänä päivänä.

Vaisman ja Zimányi (2014) kuvaavat tietovaraston osana laajempaa BI-kokonaisuutta. BI on yleisesti Vaismanin ja Zimányin mukaan "kokoelma metodeja, prosesseja, arkkitehtuureja sekä teknologioita, jotka muokkaavat raa'asta datasta merkityksellistä ja hyödyllistä tietoa päätöksenteon tueksi". Tällaisten järjestelmien he mainitsevat usein sisältävän valtavan määrän dataa erilaisista organisaation lähdejärjestelmistä, kuten ERP- ja HR-järjestelmistä. Se-

nin ja Sinhan (2005) mukaan tietovarastointia kuvaili korkeimman prioriteetin projektiksi vuosituhanteen vaihteen jälkeen yli puolet IT-johtajista.

2.2 Tiedon mallinnus

Simsion ja Witt (2004, s. 3-7) kuvaavat tiedon mallinnuksen prosessina, jossa määritellään tietokannan rakenne eli kuvaus siitä, mitä tietoa tallennetaan ja kuinka se on järjestetty. Heidän mukaansa tiedon mallinnus on ennen kaikkea suunnitteluaktiviteetti, minkä vuoksi siinä ei ole helposti määritettävissä oikeaa ratkaisua, vaikkakin useita vääriä ratkaisuja on helppo tunnistaa. Tietomallin suunnittelun he mainitsevat olennaisena niin operatiivisten tietokantojen (s. 10) kuin tietovarastoinnissa käytettävien tietokantojen (s. 27) suorituskykyä sekä hyödyllisyyttä. Tutkijoiden tai tietovarastojen kehittäjien keskuudessa ei ole selkeää yksimielisyyttä parhaasta tyylistä tietovarastoinnissa käytettävään tiedon mallinnukseen, ja sopivan tietomallin suunnittelu onkin olennaisessa osassa onnistuneen tietovarastoprojektin toteutuksessa (Jukic 2006).

Tietovarastoinnissa tiedon mallinnus jakaantuu usein kolmelle eri tasolle: konseptimalli, looginen malli ja fyysinen malli (esimerkiksi Peralta, Illarze ja Ruggia 2003; Rifaie ym. 2008; Breslin 2004). Konseptimallissa kuvataan malliin kuuluvat käsitealueet ja niihin kuuluvat entiteetit korkealla tasolla, jolla käyttäjät yleensä niitä ymmärtävät (Peralta, Illarze ja Ruggia 2003). Konseptimalli kuvaa myös käsitteiden yhteyksiä, ja sitä kutsutaankin joskus myös nimellä *Entity Relationship Model*, ERD (Rifaie ym. 2008). Looginen malli laajentaa konseptimallia tarkemmalle tasolle ja ottaa jo jossain määrin kantaa tietokanta-alustan tyyppiin, kun taas fyysinen malli kuvaa varsinaisen tietokantatoteutuksen avaimineen ja kenttineen jollekin tietokannanhallintajärjestelmälle toteutettuna (Rifaie ym. 2008). Tässä luvussa käsitellyt mallinnustekniikat kohdistuvat fyysiseen malliin.

Kuten edellä luvussa 2.1 todettiin, tietovaraston tarkoitus on tukea liikkeenjohdon päätöksentekoa ja se on aihesuuntautunut, eli se koostuu jonkin liiketoiminnallisen aihepiirin käsitteistä. Tietovaraston mallinnuksessa pyritäänkin usein mallintamaan liiketoimintaprosesseja, jotta se vastaa tavoitteeseensa tukea liiketoiminnan päätöksentekoa (Linstedt ja Olschimke 2016, s. 90). Liiketoimintaprosessien mallintamisessa oleellista on tunnistaa niihin liittyvät

käsitteet niin, että dataa voidaan tehokkaasti analysoida (Kimball ja Ross 2011, s. 14). Lähdejärjestelmissä yksittäinen asiakas voitaisiin tunnistaa esimerkiksi aluekoodin ja asiakasnumeron yhdistelmällä, ja kun tämä *avain* tallennetaan varsinaisen analysoitavan datan kanssa, voidaan tietoa ryhmitellä tehokkaasti (Linstedt ja Olschimke 2016, s. 91). Mallissa on siis olennaista tunnistaa ja tallentaa jokaisen käsitteen osalta sen *liiketoiminta-avain* (Business Key), jota voidaan joskus kutsua myös entiteetin *luonnolliseksi avaimeksi* (Linstedt ja Olschimke 2016; Kimball ja Ross 2011, s. 46).

2.2.1 Dimensionaalinen malli

Dimensionaalisen mallinnuksen toi laajalti tunnetuksi alun perin Ralph Kimball teoksessaan *The data warehouse toolkit: the complete guide to dimensional modeling,* jonka ensimmäinen versio julkaistiin vuonna 1996. Dimensionaalinen malli, jota kutsutaan usein myös tähtimalliksi (esimerkiksi Subotic, Jovanovic ja Poscic 2014) tai Kimballin malliksi (esimerkiksi Breslin 2004) on alusta alkaen tietovarastointiin suunniteltu mallinnustyyli, joka eroaa merkittävästi perinteisistä operatiivissa tietokannoissa käytetyistä relaatiomalleista, kuten kolmannesta normaalimuodosta. Esimerkiksi Breslin (2004) huomauttaa dimensionaalisen mallin rikkovan perinteisiä normalisointisääntöjä, sillä taulut sisältävät usein toistuvaa dataa. Normalisointisääntöjen rikkomisen taustalla on kuitenkin Breslinin mukaan parempi suorituskyky tietovarastoinnin näkökulmasta. Kimballin dimensionaalinen malli on kirjallisuudessa usein esitetty yhtenä yleisimmistä tietovarastointimenetelmistä, ja sen käyttö tietovarastoinnin esityskerroksena on lähes universaalia (esimerkiksi Ariyachandra ja Watson 2010; Breslin 2004; Alsqour, Matouk ja Owoc 2012). Dimensionaalinen tietomalli koostuu kahdesta taulutyypistä: *faktoista* ja *dimensioista*. Seuraavissa kahdessa kappaleessa esitellään faktataulujen ja dimensiotaulujen ominaisuuksia Kimballin ja Rossin (2011) esittämien konseptien mukaan.

Faktatauluihin tallennetaan tietoa johonkin liiketoimintaprosessiin liittyvistä mitattavista suureista. Useissa tapauksissa faktataulut ovat rivimääriltään tietovaraston suurimpia tauluja, mutta niissä on verrattain vähän sarakkeita. Yksittäinen faktataulun rivi vastaa jotain tapahtumaa, josta on tallennettu mitattavia arvoja. Tällainen tapahtuma voi olla esimerkiksi kassajärjestelmään kirjattu myyntitapahtuma. Faktatauluun liittyy olennaisena käsitteenä

sen *grain*, eli tapahtuman tarkkuuden taso. Grain voidaan määrittää esimerkin myyntitapahtumalle olevan *yksi rivi jokaista myytyä tuotetta kohden myyntitapahtumassa*. Faktataulun kaikkien rivien tulee noudattaa tätä samaa tarkkuuden tasoa. Faktataulun rivillä on siis usein suora vastine jollekin reaalimaailman tapahtumalle. Faktat ovat useimmiten numeerisia ja summattavia, eli niiden arvoja voidaan helposti summata erilaisilla ryhmittelyillä, kuten aikaväleillä. Toisaalta fakta voi olla luonteeltaan myös ei-summattava, kuten pankkititilien saldo. Laskettavien mittareiden lisäksi faktataulut sisältävät aina viiteavaimia dimensiotauluihin, jotka sisältävät tapahtumaan liittyvää kuvaavaa dataa. (Kimball ja Ross 2011, s. 10-12).

Dimensiotaulut siis ovat faktoja täydentäviä tauluja. Päinvastoin kuin faktat, dimensiot sisältävät lähinnä tekstuaalista dataa, joka voi liittyä johonkin liiketoimintatapahtumaan. Ne ovat rivimäärältään huomattavasti faktoja pienempiä, mutta voivat sisältää paljon enemmän sarakkeita. Edellä mainittuun myyntitapahtumaan voisi liittyä esimerkiksi käsitteet *myyjä* ja *tuote*. Faktataulu sisältää vain viiteavaimen erillisiin dimensiotauluihin, jotka sisältävät varsinaisen kuvaavan datan. Dimensiotaulusta siis voi tässä tapauksessa löytyä esimerkiksi myyjän henkilötiedot ja tuotteen kuvaavat tiedot. Faktataulun viiteavaimen lisäksi dimensiotaulu sisältää sen liiketoiminta-avaimeen liittyvän datan. Dimensiotauluissa on usein mukana myös esimerkiksi hierarkioihin liittyvää dataa: tuotteella voi olla tuotemerkki, joka kuuluu johonkin kategoriaan. Tällaisessa tapauksessa tulee helposti esille aiemmin mainittu datan toisteisuus ja normalisoinnin rikkominen, kun esimerkiksi saman tuotekategorian kuvaus on useilla tuoteriveillä. Vähäisen rivimääränsä vuoksi tilan säästö normalisoimalla ei kuitenkaan yleensä vaikuta merkittävästi koko tietokannan kokoon, joten lähes aina yksinkertaisuudella ja käytettävyydellä saavutetaan parempi lopputulos. (Kimball ja Ross 2011, s. 13-15).

2.2.2 Data Vault

Data Vault on Daniel Linstedtin kehittämä tiedon mallinnusmenetelmä (esimerkiksi Krneta, Jovanovic ja Marjanovic 2014). Menetelmän etuna esimerkiksi dimensionaaliseen malliin on kirjallisuudessa (Krneta, Jovanovic ja Zoran Marjanovic 2014; Jovanovic ja Bojicic 2012; Subotic, Jovanovic ja Poscic 2014; Pankov ym. 2014) esitetty sen joustavuus mallin laajentuessa tai muuttuessa, sillä se erottaa kuvaavan datan selkeästi rakenteellisesta datasta ja muutokset toteutetaan aina lisäyksinä olevassa olevaan malliin. Mallin taustalla onkin jo

lähtökohtaisesti ajatus siitä, että tietovarastointiympäristö on jatkuvan muutoksen kohteena ja kehitys tapahtuu inkrementaalisesti (Linstedt ja Olschimke 2016, s. 90). Toisaalta esimerkiksi Pankov ym. (2014) mainitsevat mallin heikkoutena sen rakenteen optimoinnin datan tallennukseen tehokkaiden kyselyiden sijasta. Data Vault -mallissa tiedon yhteydessä tallennetaan aina metadataa, jota voidaan hyödyntää esimerkiksi datan historiamuutosen jäljittämisessä tai lähdejärjestelmän replikoinnissa (Linstedt ja Olschimke 2016, s. 283-286). Data Vault -malli koostuu kolmesta taulutyypistä: *hubeista*, *linkeistä* ja *satelliiteista* (Linstedt ja Olschimke 2016, s. 91). Seuraavissa kolmessa kappaleessa käsitellään näiden taulutyyppien ominaisuuksia Linstedtin ja Olschimken esittämien konseptien mukaan.

Linstedtin ja Olschimken (2016) mukaan tavoitteena on kuvastaa liiketoimintaa mahdollisimman tarkasti ja tukea liiketoimintaa mahdollisimman tehokkaasti vastaamalla olennaisiin liiketoimintakriteereihin kuten joustavuuteen, skaalautuvuuteen ja ketteryyteen. Edelleen he määrittelevät, että Data Vault -malli "kuvastaa liiketoimintaprosesseja ja on sidottu liiketoimintaan liiketoiminta-avaimien kautta". Liiketoiminta-avaimet ovatkin olennaisessa osassa Data Vault -mallia, ja ne tallennetaan hubitauluihin. Hubitaulu sisältää vain liiketoiminta-avaimen sekä edellä mainittuja metadatakenttiä, ilman kuvaavia attribuutteja. Hubitaulu on siis uniikki lista jotain liiketoimintakäsitettä, kuten asiakasta tai laskua kuvaavia avaimia. (Linstedt ja Olschimke 2016, s. 90-94)

Liiketoiminta-avaimella tunnistettaviin entiteetteihin liiittyy kuitenkin suurimmassa osassa tapauksista myös kuvaavia attribuutteja. Data vault -mallissa hubitauluun liittyy 1-n kappaletta satelliittitauluja. Satelliittitaulu sisältää metadatan lisäksi viittauksen hubitauluun sekä hubiin liittyvät kuvaavat attribuutit. Satelliittitaulu ei siis voi olla olemassa itsenäisenä, vaan se liittyy aina johonkin hubiin. Esimerkiksi asiakas-hubiin voisi liittyä satelliitti, joka sisältää asiakkaan yhteystiedot. Satelliittitaulut tallentavat hubeihin liittyvän historiadatan; kun johonkin hubiin liittyvän käsitteen attribuuteissa tapahtuu muutoksia, satelliittitauluun lisätään uusi rivi säilyttämällä vanha versio. Tällainen *insert only* -rakenne mahdollistaa lähdejärjestelmien replikoinnin haluttuna aikana. (Linstedt ja Olschimke 2016, s. 112-114).

Entiteettien suhteita kuvataan data vault -mallissa linkkitauluilla. Linkkitaulu voisi esimerkiksi yhdistää asiakkaat tilauksiin. Linkkitaulut eivät hubien tapaan sisällä lainkaan kuvaavia attribuutteja, vaan niiden tarkoitus on ainoastaan kuvata mallin relaatioita. Linkkitauluun liit-

tyy aina vähintään 2 viittausta hubitauluihin suhteen ollessa aina teknisesti monesta moneen, vaikka varsinainen suhde reaalimaailmassa olisi yhdestä moneen. Tämä mahdollistaa osaltaan mallin joustavuuden ja helpon muokattavuuden. (Linstedt ja Olschimke 2016, s. 101-104).

2.3 Tietovarastoarkkitehtuurit

Vaikka tietovarastot ovat viimeisen vuosikymmenen aikana muodostuneet olennaiseksi osaksi useiden yritysten päätöksenteon perustaa, parhaasta tietovarastoarkkitehtuurista ei silti ole yleisesti hyväksyttyä parasta vaihtoehtoa (Alsqour, Matouk ja Owoc 2012; Ariyachandra ja Watson 2010; Sen ja Sinha 2005; Rifaie ym. 2008). Ariyachandra ja Watson (2010) huomauttavat myös, että tietovarastoarkkitehtuurin valintaan liittyvä empiirinen, teoriapohjainen tutkimustieto on varsin puutteellista. Kaksi eniten huomiota saanutta arkkitehtuurivaihtoehtoa tietovarastoinnissa ovat Inmonin (2002) esittämä *hub and spoke* -arkkitehtuuri (joskus Corporate Information Factory, CIF) sekä Kimballin (2011) esittämä *data mart bus* -arkkitehtuuri. Edellä mainittujen lisäksi kirjallisuudessa on tunnistettu usein ainakin kolme muuta arkkitehtuuria, jotka ovat jääneet vähemmälle huomiolle: itsenäiset data martit, keskitetty arkkitehtuuri ja liitetty arkkitehtuuri (Alsqour, Matouk ja Owoc 2012). Koska hub and spoke- sekä data mart bus -arkkitehtuurit ovat toimialalla käytetyimmät arkkitehtuurit (Alsqour, Matouk ja Owoc 2012; Ariyachandra ja Watson 2010; Breslin 2004), tässä tutkielmassa keskitytään näihin kahteen arkkitehtuurien tarkastelussa.

2.3.1 Data mart bus

Data mart bus -arkkitehtuuria kutsutaan usein Kimballin arkkitehtuuriksi (esimerkiksi Breslin 2004). Kimballin arkkitehtuuri on määritetty Kimballin ja Rossin (2011, s. 19) mukaan kahdesta kerroksesta: *staging-alue* ja varsinainen tietovarasto. Staging-alueella tämän määritelmän mukaan lähdejärjestelmistä ladattua dataa muokataan tietovarastoon sopivaksi ennen kuin se ladataan varsinaisiin tietovarastorakenteisiin. Edelleen tämän määritelmän mukaan Kimballin arkkitehtuurissa tietovarasto on mallinnettu dimensionaalisella mallinnuksella. Tietovarasto koostuu useista itsenäisistä *data marteista*, eli jotain yksittäistä organisaation osaa palvelevista tietovarastorakenteista (Breslin 2004). Tässä mallissa loppukäyttäjät käsit-

televät dataa suoraan tietovarastosta, eli se on samalla myös käyttäjille näkyvä esityskerros (Kimball ja Ross 2011, s. 21).

Jotta erillisistä data marteista saataisiin koko organisaation laajuinen näkymä, Kimballin arkkitehtuurissa määritetään kaikille data marteille yhteisiä dimensioita, joita kutsutaan yhdenmukaistetuiksi dimensioiksi (conformed dimensions) (Kimball ja Ross 2011, s. 51). Yleinen esimerkki yhdenmukaistetusta dimensiosta on esimerkiksi aikadimensio, sillä aika ei eroa liiketoimintaprosessien välillä (Kimball ja Ross 2011, s. 131). Yhteiset dimensiot toteutetaan jokaiseen data martiin erikseen, mutta niiden on vastattava toisiaan tarkasti, jotta niiden avulla saavutetaan organisaation laajuinen näkymä tietovarastoon (Kimball ja Ross 2011, s. 130). Tästä rakenteesta Kimball käyttää termiä *data mart bus* tai *Enterprise Data Warehouse bus*.

2.3.2 Hub and spoke

Hub and spoke -arkkitehtuurilla tavoitellaan tilannetta, jossa kehitys tapahtuu iteratiivisesti niin, että tietovarastoon lisätään alue kerrallaan uusia osia, joista muodostetaan organisaation laajuinen näkymä datasta (Singh 2011). Linstedt ja Olschimke (2016, s. 13) kuvaavat Hub and spoke -arkkitehtuurin koostuvan kolmesta kerrokseta: staging-alue, keskitetty tietovarasto ja esityskerros. Tällaisessa arkkitehtuurissa pyritään Linstedtin ja Olschimken mukaan säilyttämään data tietovarastossa mahdollisimman atomisella tasolla, ja se on usein mallinnettu kolmannessa normaalimuodossa samaan tapaan kuin operatiivisten järjestelmien tietokannat. Atomisen tason keskitetty tietovarasto palvelee useita eri esityskerroksen data marteja, eli se on arkkitehtuurin *hubi* (Singh 2011). Esityskerros tässä arkkitehtuurissa koostuu erillisistä data marteista, jotka on usein toteutettu dimensionaalisella mallinnuksella, vaikkakaan dimensionaalisen mallin käyttö ei ole varsinaisesti arkkitehtuurin asettama vaatimus. Käyttäjät käsittelevät dataa esityskerroksen tarjoamien rakenteiden kautta, ja nämä rakenteet ovat arkkitehtuurin *spoket*, jotka ovat riippuvaisia hubista (Singh 2011; Linstedt ja Olschimke 2016).

Hub and spoke -arkkitehtuurissa siis pyritään alusta asti suunnitelemaan organisaation laajuinen ratkaisu; esityskerros koostuu useista data marteista, jotka voivat palvella erilaisia käyttäjäkuntia ja olla eri tavoin mallinnettuja, mutta käyttävät kuitenkin lähteenään keskite-

tyn tietovaraston dataa (Breslin 2004). Tällä rakenteella pyritään saamaan aikaan niin sanottu "single version of truth", eli data on sama kaikille koko organisaation halki riippumatta käytetystä esityskerroksen rakenteesta (Ariyachandra ja Watson 2010).

2.3.3 Data Vault 2.0

Data Vault 2.0 -arkkitehtuuri noudattaa pääosin rakenteeltaan samanlaista Hub and spoke -rakennetta kuin edellä luvussa 2.3.2. Linstedtin ja Olschimken (2016) mukaan se koostuu vastaavasti staging-alueesta, tietovarastosta ja esityskerroksesta, mutta se sisältää joitain omia erityispiirteitään, minkä vuoksi se on erotettu muista Hub and spoke -arkkitehtuureista. Seuraavat kolme kappaletta käsittelevät Data Vault 2.0 -arkkitehtuuria Linstedtin ja Olschimken (2016) esittämien konseptien mukaan.

Staging alueen tarkoitus on muiden arkkitehtuurien tapaan saada data ulos lähdejärjestelmästä, jotta lähdejärjestelmään kohdistuu mahdollisimman vähän kuormitusta. Eroten muista arkkitehtuureista DV2.0:ssa staging-alueella ei ole tarkoitus koskaan säilyttää historiadataa, vaan sinne ladataan aina vain kulloisenkin erän sisältämä data. Staging-alueen taulut duplikoivat lähdejärjestelmän taulut sellaisenaan lisäten ainoastaan joitain teknisiä kenttiä. Teknisiä kenttiä ovat esimerkiksi lähdejärjestelmän nimi sekä latauksen aikaleima, joita tarvitaan historiatiedon ja datan myöhemmän jäljittämisen tarpeisiin. (Linstedt ja Olschimke 2016, s. 25-26)

Selkeimpänä erona on tietovarastossa käytettävä Data Vault -mallinnustekniikka (ks. luku 2.2.2). DV2.0 korostaa tietovarastoa ennen kaikkea *datan* tallentamiseen. Tällä tarkoitetaan sitä, että datan käsittely ennen sen lataamista tietovarastoon pyritään pitämään minimissä, jolloin liiketoimintaan liittyvät säännöt esimerkiksi datan koostamisesta tai puhdistamisesta käsitellään myöhemmin esityskerroksessa. Tietovarasto onkin jaettu usein loogisiin *raw vault* ja *business vault* -kerroksiin. Raw vault sisältää datan sellaisena kuin se lähdejärjestelmästä saapuu ja mahdollistaa näin ollen lähdejärjestelmän replikoinnin ja datan jäljittämisen alkuperäiseen lähteeseen. (Linstedt ja Olschimke 2016, s. 26-27)

Esityskerroksesta käytetään DV2.0:ssa nimeä *information mart*, jolla korostetaan sen roolia datan muuttamisessa tiedoksi. Esityskerros voi koostua esimerkiksi Data Vault -mallinnetusta

Business Vault -kerroksesta sekä dimensionaalisesti mallinnetuista rakenteista. Information mart on ainoa kerros, johon loppukäyttäjillä on DV2.0 arkkitehtuurissa suora pääsy. Esityskerroksessa data on siis varsinaisesta tietovarastokerroksesta poiketen yleensä eri tavoin käsiteltyä, muokattua tai puhdistettua. (Linstedt ja Olschimke 2016, s 27).

2.4 Tietovaraston populointi

Luvussa 2.1 määritettiin tietovaraston sisältävän usean eri operatiivisen järjestelmän datasta muodostetun kokoelman dataa. Tietovaraston populointi lähdejärjestelmien datalla on perinteisesti suoritettu ETL (Extract, Transform, Load) -prosesseilla (Castellanos ym. 2009). Tällaisessa prosessissa data ladataan ensin sellaisenaan staging-alueelle (Extract), minkä jälkeen dataa muokataan erilaisilla transformaatioilla (Transform), ja lopuksi muokattu data ladataan tietovarastorakenteisiin (Load) (Breslin 2004). ETL-prosesseilla siis muunnetaan erilaisten lähteiden data tietovaraston skeemaa vastaavaksi ja populoidaan tietovarastotauluja datalla. ETL-prosessit voivat olla monimutkaisia ja niiden kehitys vie merkittävän osan tietovarastokehitykseen käytetystä ajasta (El-Sappagh, Hendawi ja El Bastawissy 2011).

Koska jo määritelmänsä mukaan tietovaraston tulee olla integroitu ja se koostuu usean eri lähteen datasta, extract-vaiheessa tulee huomioida erilaisten lähteiden ominaispiirteet. Datan lataaminen operatiivisen järjestelmän tietokannasta on erilainen prosessi kuin esimerkiksi tekstitiedoston lataaminen (El-Sappagh, Hendawi ja El Bastawissy 2011). Kakishin ja Kraftin (2012) mukaan datan lataaminen useista erilaisista lähteistä on usein koko ETL-prosessin haastavin vaihe, sillä lähdejärjestelmien data on usein monimutkaista, joten olennaisen datan määrittäminen on hankalaa. Edelleen heidän mukaansa tämän vaiheen onnistuminen määrittää seuraavien vaiheiden onnistumisen.

Transformation-vaiheen tarkoitus on muuttaa data tietovaraston rakennetta vastaavaksi. Kakish ja Kraft (2012) tuovat esiin mahdollisina datan muutoksina esimerkiksi järjestämisen tai uusien arvojen laskemisen. Koska useissa tapauksissa tietovarastossa data ei ole samalla tarkkuuden tasolla kuin lähteessä, tässä vaiheessa voidaan myös muuttaa tiedon tarkkuuden tasoa (El-Sappagh, Hendawi ja El Bastawissy 2011). Tämä vaihe voi olla hyvin yksinkertainen tai sisältää paljon erilaisia toimenpiteitä, sillä datan lähteiden transformaatiotarpeet

vaihtelevat suuresti (Kakish ja Kraft 2012).

Load-vaiheessa populoidaan tietovarastotaulut edeltävien vaiheiden perusteella muodostuneella datalla, ja vaaditut operaatiot vaihtelevat käytettävän tietovarastoarkkitehtuurin ja organisaation vaatimusten mukaan (Kakish ja Kraft 2012). Esimerkiksi Kimballin arkkitehtuurissa ei välttämättä säilytetä dimensiotaulujen historiaa, jolloin olemassaolevaa riviä voidaan päivittää. Toisaalta taas Data Vault -arkkitehtuurissa pyritään siihen, että dataa ei päivitetä koskaan, vaan datan päivitys tapahtuu lisäämällä tietokantaan uusi rivi.

Transform- ja Load-vaiheiden järjestys voi myös olla päinvastainen kuin edellä esitetty. Tällöin puhutaan ELT-prosesseista (Extract, Load, Transform). Tässä lähestymistavassa data ladataan tietovaraston staging-alueelle tekemättä transformaatioita, ja transformaatiot tehdään vasta ennen varsinaisiin tietovarastorakenteisiin lataamista. (Puonti ym. 2016).

ETL-prosessien suunnittelussa ja toteutuksessa olennaisessa osassa on tietovirtojen määritys, eli taulun kenttätasolla esitetty kuvaus, mistä lähdetaulun kentistä kohdetauluun ladataan dataa (Simitsis 2003). Tätä määritystä kutsutaan usein *mappingiksi* (esimerkiksi Simitsis 2003; Puonti ym. 2016). Lisäksi ETL-prosessissa tulee määrittää latauksessa toteutettavat transformaatiot. Tietovaraston ETL-latauksien toteuttamiseen on ollut jo pitkään saatavilla työkaluja, ja vaikka ne helpottavat prosessien toteutusta, ne vaativat silti paljon manuaalista työtä esimerkiksi transformaatioiden määrityksessä (Puonti ym. 2016; Simitsis, Vassiliadis ja Sellis 2005).

3 Tietovarastokehitys ja automatisointi

Tässä luvussa käsitellään tietovarastokehityksen työvaiheita ja niihin liittyviä toimenpiteitä. Lisäksi tarkastellaan, missä vaiheissa ja toimenpiteissä automatisointia voidaan hyödyntää ja millaisia etuja sillä voidaan saavuttaa.

Rahman ja Rutz (2015) kuvaavat tietovarastokehitystä työlääksi ja aikaavieväksi prosessiksi, jota voidaan kuitenkin kiihdyttää samalla parantaen laatua, kun hyödynnetään DWA (Data Warehouse Automation) -ohjelmistoja. Tällaisilla ohjelmistoilla voidaan automatisoida muun muassa tiedon mallinnusta, skeeman luontia, datan latausten kartoituksia sekä ETL-kehitystä.

Tietovaraston kehitykseen kuuluu useita selkeitä vaiheita: ennen kehitystä tapahtuvat aktiviteetit, arkkitehtuurin valinta, skeeman luonti, toteutus ja ylläpito. Näistä skeeman luonti ja toteutus ovat alueita, jotka perinteisesti ovat sisältäneet paljon manuaalista työtä, jota voidaan tehostaa automatisoimalla (Tomingas, Kliimask ja Tammet 2015; Phipps ja Davis 2002). Tässä tutkielmassa painopisteenä on tietovarastokehityksen automatisointi, joten tarkastelun ulkopuolelle jätetään aktiviteetit, joita ei yleensä voida automatisoida. Tällaisia ovat ennen kehitystä tapahtuvat aktiviteetit kuten liiketoiminnan vaatimusten määrittely sekä arkkitehtuurin valinta. Ylläpito taas laajempana käsitteenä sisältää aktiviteetteja liittyen skeeman luontiin ja tietovaraston toteutukseen, joten sitä ei käsitellä erillisenä aihealueenaan. Skeeman luonti liittyy vahvasti tiedon mallinnukseen (ks. luku 2.2) ja tietovaraston populointi ETL/ELT -prosesseihin (ks. luku 2.4).

Tietovarastokehityksen toteutusvaihe sisältää useita eri aktiviteetteja kuten lähdemääritykset, datan lataaminen ja loppukäyttäjille tarjottavien applikaatioiden toteutus. Kaikkiin näihin aktiviteetteihin liittyy olennaisesti metadatan hallinta (Sen ja Sinha 2005). Myös Krneta, Jovanovic ja Zoran Marjanovic (2014) sekä Chaudhuri ja Dayal (1997) nostavat metadatan tärkeyden esille, sillä sitä tarvitaan kuvaamaan, mitä dataa tietovarasto sisältää ja missä se sijaitsee. Rifaie ym. (2008) huomauttavat metadatan tallentamisen olevan tärkeää edeltävän lisäksi myös muutoksenhallinnassa. Heidän mukaansa datan laatua tulee tarkkailla omistajien toimesta. Kuitenkin Pankovin ym. (2014) mukaan useat organisaatiot eivät käsittele meta-

dataa järjestelmällisesti johtuen esimerkiksi puuttuvista standardeista, keskitetyn metadatan hallinnan tuomasta työmäärästä sekä metadatan hallinnan hyötyjen puuttellisesta ymmärtämisestä. Metadatalla tarkoitetaan tässä yhteydessä pääosin teknistä, tietovaraston toteutusta kuvaavaa dataa, kuten tietovarastotaulujen kenttien tietotyyppejä tai latausten mapping-määrityksiä.

Tietovarastokehityksessä on Puonnin ym. 2016 mukaan käytetty eri tarkoituksiin eri työkaluja, esimerkiksi mallintamiseen ja ETL-latausten toteuttamiseen voi olla käytössä omat työkalunsa. Puonti ym. keskittyivät Data Vault -tietovaraston automatisointiin, sillä sen laajennettavuus ja joustavuus ovat heidän mukaansa johtaneet automatisoinnin tarpeeseen, kun nämä ominaisuudet ovat samalla kasvattaneet populointiin vaadittavien latausoperaatioiden määrää. Toisaalta he totetavat mallin tukevan hyvin automatisointia ja suuren osan objekteista olevan helposti generoitavissa automaattisesti niiden standardimuotoisuuden vuoksi. Samalla saavutetaan heidän mukaansa helposti yhteneväiset nimeämiskäytännöt sekä arkkitehtuuriratkaisut, jolloin ylläpito ja jatkokehitys on helpompaa, kun automatisoimalla minimoidaan riski inhimillisille virheille manuaalisissa prosesseissa. Puonti ym. esittivät metadataan pohjautuvan menetelmän, jolla voidaan automaattisesti generoida tietokantaobjektien luontilauseita sekä ETL-latauksien koodia.

Myös Pankov ym. (2014) pitävät metadataa ja sen järjestelmällistä hallintaa olennaisena tietovarastokehityksen automatisoinnissa, koska sen avulla voidaan automaattisesti generoida ETL-prosesseja, joiden toteutus voi viedä jopa yli 50% koko tietovarastoprojektin työajasta. Metadatan hyödyntämiseen perustuvaa kehitystä he kuvaavat *metadata-driven* -kehitykseksi. Metadataa voidaan Pankovin ym. mukaan hyödyntää tehokkaimmin, kun se tallennetaan mahdollisimman geneerisessä muodossa. Geneerinen tallennusmuoto toisaalta tekee datan käsittelystä vaativaa, minkä vuoksi siihen tarvitaan erillinen työkalu. Sen ja Sinha (2005) nostavat esille myös metadatan suuresta määrästä johtuvan tarpeen erilliselle työkalulle metadatan hallintaan. Tällaisen työkalun ja etukäteen määritettyjen ETL-prosessipohjien avulla voidaan kehitystä tehostaa merkittävästi.

Tomingas, Kliimask ja Tammet 2015 pitivät yhtä lailla tietovarastolatauksiin liittyviin kenttäkartoituksiin ja monimutkaisiin SQL-skripteihin liittyvää manuaalista toteutustyötä virhealttiina ja aikaavievänä, minkä vuoksi sitä pitäisi heidän mukaansa pyrkiä automatisoimaan

valmiiden mallien pohjalta luoduilla SQL-skripteillä. Myös Jörg ja Dessloch (2008) esittävät keinoja generoida latausprosesseja automaattisesti, kun pohjalla on geneerinen määritys, mistä kohdetauluun ladattava data on peräisin. Jörg ja Dessloch huomauttavat tällaisen automatisoinnin olevan hyödyksi erityisesti, kun dataa ladataan inkrementaalisesti ja latausprosessi on tavallista suoraa latausta monimutkaisempi.

Petrović ym. (2017) pitävät ETL-prosessien kehityksen automaatiossa tärkeänä, että koodin generoimisen lisäksi työkalu mahdollistaa ETL-prosessien formaalin määrittelyn. Toisaalta nämä vaatimukset tukevat toisiaan, sillä formaali määrittely mahdollistaa heidän mukaansa tehokkaan koodin generoimisen ilman, että generoitua koodia tarvitsee manuaalisesti muokata. Tehokkaan automaatiotyökalun tulisi tukea Petrovićin ym. lisäksi myös muita kehitykseen liittyviä toimenpiteitä. Kehityksen aikana voi olla esimerkiksi tarve suorittaa latauksia dynaamisesti, jolloin voidaan tehokkaasti vastata muuttuviin tarpeisiin. Lisäksi lataukset tulee voida helposti siirtää ajoalustoille ja työkalun tulee tukea latausten versiointia.

Vaikka ETL-prosesseja ja niiden toteutusta pidetäänkin usein tietovarastoprojektin työläimpänä vaiheena, kehitystä voidaan automatisoida myös muilla osa-alueilla. Phipps ja Davis (2002) esittivät algoritmin, jonka avulla voidaan automaattisesti luoda tietovaraston konseptimalleja lähdejärjestelmän perusteella. Tällaisella automaatiolla voidaan heidän mukaansa helpottaa etenkin kehityksen alkuvaiheita, kun pyritään kuvaamaan korkean tason tietomalleja ennen fyysisten mallien toteutusta. Automaatiota voidaan hyödyntää myös konseptitason mallin muuntamisessa tarkemman tason loogiseksi malliksi. Tähän tarkoitukseen Peralta, Illarze ja Ruggia (2003) kehittivät sääntöihin perustuvan viitekehyksen, jolla automatisoidaan useita loogisen mallin muodostamiseen liittyviä päätöksiä. Aadil ym. (2016) kehittivät menetelmän, jolla voidaan luoda dimensionaalisia tietovarastomalleja automaattisesti hyödyntäen SQL-koodista generoituja ontologioita.

Kuten luvuissa 2.2.2 ja 2.3.3 on mainittu, Data Vault -tietovarasto tarvitsee aina tehokasta datan analysointia varten esityskerroksen. Ylimääräinen esityskerros ei kuitenkaan välttämättä johda merkittävästi suurempaan työmäärään, sillä esityskerroksen luomista voidaan myös automatisoida. Krneta, Jovanovic ja Marjanovic (2016) sekä Golfarelli, Graziani ja Rizzi (2016) esittivät menetelmiä, joilla Data Vault -mallinnetusta tietovarastosta voidaan automaattisesti generoida esityskerroksen dimensionaalisia tietomalleja. Molemmat mene-

telmät hyödyntävät metadataa ja Data Vault -mallin luontaisia ominaisuuksia automaattisten dimensionaalisten mallien muodostamisessa.

Tietovarastoprosessia voidaan siis jossain määrin automatisoida kehitysvaiheen alusta loppuun saakka, kun automaatiota voidaan hyödyntää niin mallinnuksessa kuin varsinaisessa datan lataamisessa. Castellanos ym. (2009) toteavat lisäksi, että tietovarastolatausten automaattisessa luomisessa on myös ongelmia, sillä on hankala kehittää tähän metodi, joka toimii kaikissa ympäristöissä. Näin ollen sen edut tulevat Castellanoksen ym. mukaan parhaiten esiin vain, kun tietovarastoalusta, tietomalli sekä lähdejärjestelmä tukevat tällaisia metodeja. Toisaalta Petrovicin ym. (2017) mukaan tietovarastokehityksen, etenkin ETL-prosessien tehokkaassa automaatiossa on olennaista, että automaatioväline on suunnattu johonkin tiettyyn ympäristöön.

4 Tutkimusmenetelmät

Tässä luvussa käsitellään tutkimusmenetelmiä, joita tutkielman toteuttamisessa noudatettiin. Tutkimus toteutettiin suunnittelutieteen näkökulmasta suunnittelutieteellisenä tutkimuksena. Luvussa 4.1 käsitellään yleisesti suunnittelutiedettä ja suunnittelutieteellistä tutkimusta. Luvussa 4.2 tarkastellaan suunnittelutieteellisen tutkimuksen arviointia. Luvussa 4.3 perehdytään tarkemmin edellä mainittujen menetelmien soveltamiseen tässä tutkielmassa sekä esitellään tutkimusprosessia.

4.1 Suunnittelutiede ja tutkimus

Hevner ym. (2004) määrittivät suunnittelutieteellisen tutkimuksen toteuttamiselle 7 ohjenuoraa (ks. Taulukko 1). Näistä tärkeimpänä esimerkiksi Peffers ym. (2007) pitivät sitä, että tutkimuksen tulee tuottaa "artefakti, joka on luotu vastaamaan ongelmaan". Lisäksi Hevnerin ym .(2004) mukaan artefaktin tulee olla vastaus ennalta ratkaisemattomaan ja tärkeään liiketoimintaongelmaan. Artefaktin kehityksen tulisi heidän mukaansa olla etsintäprosessi, joka hyödyntää olemassaolevaa teoriaa ja tietoutta päätyäkseen määritellyn ongelman ratkaisuun.

Suunnittelutieteellinen tutkimus eroaa monin paikoin perinteisestä luonnollisesta tutkimuksesta. Esimerkiksi Marchin ja Smithin (1995) mukaan luonnollinen tutkimus pyrkii ymmärtämään todellisuutta, kun taas suunnittelutieteellinen tutkimus pyrkii tuottamaan asioita, joilla on jokin inhimillinen tarkoitus. Edelleen he toteavat suunnittelutieteellisen tutkimuksen olevan luonteeltaan teknologiapainotteista, ja sen tuotoksia arvioidaan arvoa tai hyödyllisyyttä mittaavia kriteereitä vasten.

Peffers ym. (2007) kuvaavat tietojärjestelmien tutkimusta sovellettuna tieteenhaarana, sillä usein siinä sovelletaan muista tieteenhaaroista, kuten taloustieteestä tai tietojenkäsittelytieteestä peräisin olevaa teoriaa. Suunnitteluaktiviteetit ovat keskeisiä useimmissa sovelletuissa tieteissä, ja suunnittelun tutkimuksella on pitkät perinteet monissa tieteenhaaroissa (Hevner ja Chatterjee 2010).

Taulukko 1: Suunnittelutieteellisen tutkimuksen ohjenuorat. Suomennettu Hevnerin ja Chatterjeen 2010 taulukosta 2.1.

Ohjenuora	Kuvaus
1. Suunnittelu artefaktina	Suunnittelutieteellisen tutkimuksen tulee tuottaa toteuttamiskelpoinen artefakti konstruktion, mallin, metodin tai ilmentymän muodossa.
2. Ongelman merkityksellisyys	Suunnittelutieteellisen tutkimuksen tavoite on kehittää teknologiapainotteisia ratkaisuja tärkeisiin ja olennaisiin liiketoimintaongelmiin
3. Suunnittelun arviointi	Suunnitteluartefaktin käytännöllisyys, laatu ja tehokkuus tulee arvioida tarkasti hyvin toteutetuilla arviointimenetelmillä
4. Tutkimuksen kontribuutio	Tehokkaan suunnittelutieteellisen tutkimuksen tulee tuottaa selkeitä ja verifioitavia kontribuutioita
5. Tutkimuksen täsmällisyys	Suunnittelutieteellinen tutkimus nojaa täsmällisten metodien käyttöön niin artefaktin luomisessa kuin arvioimisessa
6. Suunnittelu etsintäprosessina	Tehokkaan artefaktin etsintä vaatii käytössä olevien keinojen hyödyntämistä halutun lopputuloksen saavuttamiseksi
7. Tutkimuksen kommunikointi	Suunnittelutieteellinen tutkimus tulee esittää tehokkaasti niin teknologiasuuntautuneelle kuin hallintosuuntautuneelle yleisölle

March ja Smith (1995) kuvaavat suunnittelutieteen koostuvan kahdesta olennaisesta osiosta; rakentaminen ja arviointi. Rakentamisvaiheen he määrittävät prosessiksi, jonka tarkoituksena on muodostaa artefakti jotain määrättyä tarkoitusta varten. Arviointivaiheessa puolestaan arvioidaan, kuinka hyvin artefakti suoriutuu tarkoituksesta, jota varten se on rakennettu. Rakentamisvaiheessa keskeinen ongelma heidän mukaansa on se, että artefaktin suorituskyky riippuu ympäristöstä, jossa se toimii. Ympäristöä he pitävät olennaisena tekijänä, sillä puut-

teellinen ymmärrys ympäristöstä voi johtaa ei-toivottuihin sivuvaikutuksiin. Artefaktin ympäristön he totevat liitttyvän myös arviointivaiheeseen, sillä eri ympäristöissä suorituskykyä voidaan mitata eri tavoin. Tutkimuksessa onkin siis syytä artefaktin arvioinnin lisäksi kohdistaa arviointikriteetit artefaktiin tietyssä ympäristössä.

Peffers ym. (2007) muodostivat vahvasti Hevnerin ym. 2004 määrittelemien periaatteiden pohjalta suunnittelutieteellisen tutkimuksen toteuttamiseksi kuusivaiheisen prosessin. Prosessin vaiheet on määritetty taulukossa 2.

Taulukko 2: Peffersin ym. 2007 prosessi suunnittelutieteellisen tutkimuksen toteuttamiseen

Vaihe	Kuvaus
1.	Ongelman tunnistaminen ja motivaatio
2.	Ratkaisun tavoitteiden määritys
3.	Suunnittelu ja kehitys
4.	Demonstraatio
5.	Arviointi
6.	Kommunikointi

Tätä prosessia pidetään hyvin vakiintuneena suunnittelutieteellisen tutkimuksen toteuttamisessa. Toisaalta esimerkiksi Gregor ja Hevner (2013) sekä Conboy, Gleasure ja Cullina (2015) huomauttavat, ettei prosessia ole välttämättä tarpeen seurata täysin sellaisenaan, vaikka se tarjoaakin hyvän rungon suunnittelutieteelliselle tutkimukselle. Conboy, Gleasure ja Cullina (2015) esittivät mallista laajennetun version, jossa hyödynnetään ohjelmistokehityksestä tuttuja ketteriä menetelmiä, jolloin prosessi voi alkaa myös muista vaiheista kuin ensimmäisestä ja toisaalta sen ei tarvitse edetä järjestelmällisesti vaan useina iteraatioina.

Lee, Thomas ja Baskerville (2015) tuovat esille sen, että suunnittelutieteellinen tutkimus on pitkään painottanut teknologiasuuntautuneita IT-artefakteja. Heidän näkemyksensä mukaan suunnittelutieteessä tulisi kiinnittää enemmän huomiota siihen, että tärkeydestään huolimatta IT-artefaktit ovat kuitenkin lähes poikkeuksetta osa jotain laajempaa kontekstia, järjestelmää. Järjestelmä taas voi olla suurempi kuin osiensa summa, sisältäen teknologian lisäksi sosiaalisen artefaktin. Sosiaalisen artefaktin he määrittävät artefaktiksi, joka "koostuu tai si-

sällyttää suhteita tai vuorovaikutusta yksilöiden välillä tai keskuudessa, jonka avulla yksilö pyrkii ratkaisemaan ongelmansa, saavuttamaan maalinsa tai palvelemaan hänen tarkoituksiaan". Teknologia-artefaktin ja sosiaalisen artefaktin lisäksi Lee, Thomas ja Baskerville (2015) määrittävät laajemman *järjestelmäartefaktin* sisältävän myös informaatioartefaktin.

Myös Leoz ja Petter (2018) painottavat artefaktien sosiaalisia aspekteja. Toteutetulla artefaktilla voi olla vaikutuksia yksilöiden välisiin suhteisiin ja vuorovaikutukseen, minkä vuoksi sosiaaliset aspektit tulisi huomioida artefaktin kehityksessä ja arvioinnissa. He esittävät Leen, Thomasin ja Baskervillen 2015 tapaan mielekkääksi kehityksen kohteeksi laajempaa järjestelmäartefaktia teknologiaan painottuvan artefaktin sijaan. Yleistä Leozin ja Petterin mukaan on, että artefaktin mahdolliset vaikutukset sosiaalisessa kontekstissa huomioidaan ennen toteutusta, mutta toteutusvaiheessa painopiste siirtyy vain tekniseen näkökulmaan, minkä jälkeen arvioinnissa sosiaaliset osiot jäävät vähemmälle huomiolle. Koska toteutettu artefakti voi vaikuttaa sosiaalisiin aspekteihin, tulisi ne Leozin ja Petterin mukaan huomioida koko kehityksen ajan.

Baskervillen ym. (2018) mukaan suunnittelutieteellisen tutkimuksen tavoitteiden, ja siten myös tavoitelluiden kontribuutioiden voidaan katsoa olevan luonteeltaan teknologisia tai tieteellisiä. Useissa tapauksissa artefaktin kehitystä ja teknologista näkökulmaa on heidän mukaansa painotettu enemmän, mutta usein suunnittelutieteellisen tutkimuksen odotetaan tuottavan myös jossain määrin teoreettisia tuloksia. Toisaalta he kuitenkin huomauttavat, että ennen näkemättömän artefaktin kuvaus voidaan nähdä kontribuutiona suunnitelutieteelle, joten teknologisten ja teoreettisten kontribuutioiden raja ei aina ole täysin yksiselitteinen.

4.2 Artefaktin arvioiminen

Artefaktin arvioimista pidetään yleisesti hyvin tärkeänä osana suunnittelutieteellistä tutkimusta. Arvioinnin tärkeyttä korostavat esimerkiksi March ja Smith (1995), jotka nostivat arvioinnin toiseksi pääosaksi suunnittelutieteellistä prosessia. Hevner ym. (2004) korostavat suunnittelutieteellisen tieteellisen tutkimuksen ohjenuorissaan tarvetta osoittaa artefaktin hyödyllisyys hyvin toteutetuilla arviointimenetelmillä. Kuechlerin ja Vaishnavin 2011 mukaan arviointi "varmistaa tutkimuskonseptin kehityksen yksinkertaista prototyyppiä pi-

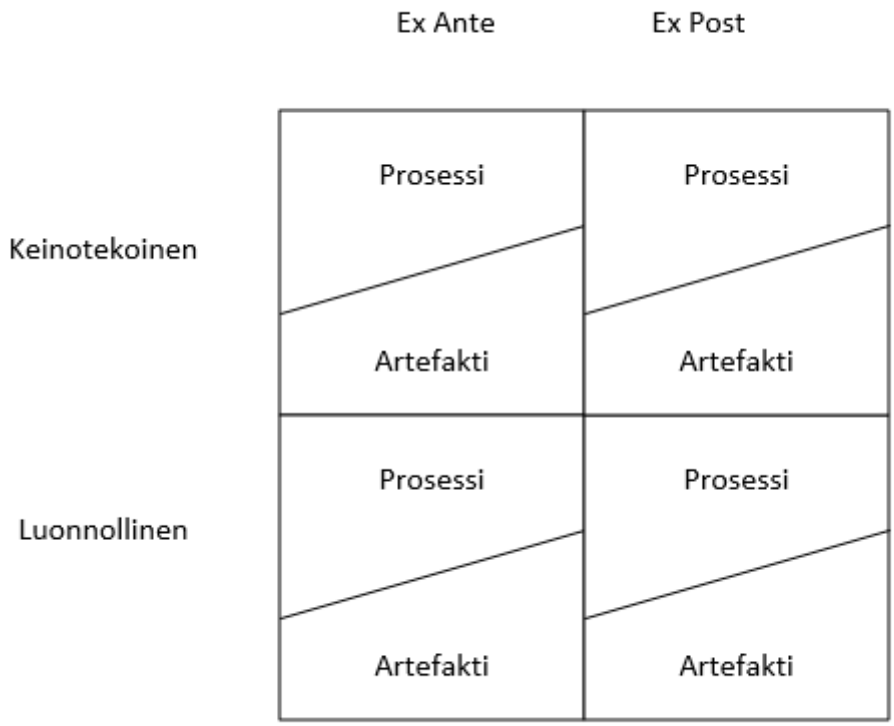
demmälle", ja laadukkaan tutkimuksen raportointi vaatii vertailua parhaiden edeltävien ratkaisujen kanssa tai kenttäarviointia oikean maailman olosuhteissa. Pries-Heje, Baskerville ja Venable (2008) kuitenkin huomauttavat, että edellä mainitussa kirjallisuudessa ei kuitenkaan juurikaan tarjota ohjeistusta suunnittelutieteellisen tutkimuksen arvioinnin metodeihin tai strategioihin.

Klecun ja Cornford (2005) toteavat yleisesti tietojärjestelmien arvionnista, että erilaisilla viiteryhmillä on erilaisia odotuksia, minkä vuoksi "on epäselvää, kuinka onnistumista tai hyötyjä voidaan arvioida, tai mitä onnistuminen ylipäätään tarkoittaa". Näin ollen arvioinnissa ei tulisi keskittyä vain onnistumiseen. Heidän mukaansa arvioinnissa tulee huomioida arvionnin konteksti (kuka arvioi ja miksi), prosessi (kuinka arviointi tapahtuu) sekä sisältö (mitä arvioidaan).

Arvionti on usein suunnittelutieteen kirjallisuudessa sijoitettu suunnittelun ja toteutuksen jälkeiseksi vaiheeksi (Pries-Heje, Baskerville ja Venable 2008). Artefaktin toteutuksen jälkeistä arviointia kutsutaan Pries-Hejen, Baskervillen ja Venablen mukaan *ex post* -arvioinniksi. Kuitenkin heidän mukaansa arviointia voidaan ajatella tapahtuvan laajalti jo ennen varsinaista rakennusvaihetta. Artefaktin ollessa fyysinen objekti kuten sovellus, artefaktia ja sen toteuttamisessa käytettäviä teknologioita arvioidaan heidän mukaansa väistämättä ongelman tunnistamisessa ja ratkaisun tavoitteiden määrittämisessä. Ennen toteutusta tehtävää arviointia he kutsutvat *ex ante* -arvioinniksi, ja sillä voidaan esimerkiksi pyrkiä selvittämään, onko artefaktin kehittäminen kannattavaa.

Sonnenberg ja Brocke (2012) kohdistivat erityistä huomiota ex ante -arviointiin, sillä heidän mielestään täsmällisen suunnittelutieteellisen tutkimuksen saavuttamiseksi ei riitä, että artefakti kehitetään ja arvioidaan vain sen käyttöä tai hyödyllisyyttä. Tästä syystä he esittävät, että arviointia tulisi tehdä prosessin aikana jatkuvasti, jotta tutkija voi tehdä johdonmukaisia suunnittelupäätöksiä toimivan ja hyödyllisen artefaktin toteuttamiseksi. Näin ollen arviointikertoja tulisi siis tapahtua yksittäisen syklin aikana useaan kertaan.

Pries-Heje, Baskerville ja Venable (2008) esittivät suunnittelutieteellisen tutkimuksen evaluointiin viitekehyksen (kuvio 1), joka jakautuu luonnollisiin ja keinotekoisiin menetelmiin. Luonnollisissa menetelmissä arvioidaan varsinaista artefaktia oikeiden käyttäjien toimesta.



Kuvio 1: Viitekehys suunnittelutieteellisen tutkimuksen arviointiin. Suomennettu Pries-Heje, Baskerville ja Venable (2008) kuvio 1.

Keinotekoiset menetelmät mahdollistavat esimerkiksi suunnitteluteorioiden oikeaksi tai vääräksi osoittamisen. Kummankin kategorian menetelmia voidaan tässä viitekehyksessä soveltaa joko ex ante tai ex post. Lisäksi viitekehyksen mukaan arvioinnissa tulee määrittää, onko arvioinnin kohteena prosessi vai artefakti.

Venable, Pries-Heje ja Baskerville (2012) laajensivat viitekehystään myöhemmin, sillä se keskittyi vain erilaisten arviointikohteiden tunnistamiseen. Se siis vastasi hyvin kysymyksiin siitä, mitä suunnittelutieteellisen tutkimuksen arvioinnissa pitää huomioida, mutta ei tarjonnut ohjausta siihen, kuinka valinnat varsinaisesti tehdään. Laajennetussa versiossa he tarjosivat tarkemman ohjeistuksen sille, mihin viitekehyksen osioihin arviointia tulisi kohdistaa. Ohjeistus esimerkiksi tarjoaa paremman vertailupohjan ex ante ja ex post -arvioinnin sekä luonnollisten ja keinotekoisten menetelmien valinnassa. Esimerkiksi ex ante -menetelmät sopivat paremmin suunnitteluun tai prototyypin arviointiin, kun taas ex post sopii parem-

min valmiimman tuotteen arviointiin. Tämän lisäksi laajennettu malli antaa konkreettisia esimerkkejä eri arviointikohteille sopivista menetelmistä. Esimerkiksi fokusryhmäarviointi luonnollisena menetelmänä sopii toteutettavaksi sekä ex ante että ex post.

Koska artefakti on usein suunniteltu toimimaan jossain ympäristössä ja sosiaalisessa kontekstissa, myös arvioinnissa tulisi huomioida artefaktin vaikutuksia ympäristössään. Artefaktin onnistumisen arviointiin liittyy vahvasti, kuinka se soveltuu suunniteltuun ympäristöönsä. Hyvin sosiaaliset vaikutukset ja ympäristön huomioineen tutkimuksen tuottama artefakti voi parhaimmillaan loistaa tilanteessa, jossa artefakti sopii hyvin ympäristöönsä. Toisessa ääripäässä taas artefaktin voidaan arvioida jäävän kokonaan käyttämättä, mikäli sosiaalisia aspekteja ei kehityksessä huomioitu riittävästi ja artefakti ei sovellu ympäristöönsä. Näiden ääripään väliin jäävät vielä vaihtoehdot, joissa artefakti selviytyy joko sosiaalisen rakenteen mukautuessa artefaktiin tai artefaktin mukautuessa ympäristöön. (Leoz ja Petter 2018).

4.3 Menetelmien soveltaminen tässä tutkielmassa

Tutkimuksessa toteutettiin suunnitteluartefaktina sovellus, joka tiettyjä ominaisuuksia automatisoimalla auttaa tehostamaan tietovarastokehitystä. Tutkimuksen toteutuksessa hyödynnettiin Peffersin ym. 2007 kuvaamaa kuusivaiheista prosessia (ks. Taulukko 2). Tutkimuksessa toteutettiin prosessin mukaisesti kaksi sykliä hieman eri osa-alueita painottaen, vaikkakaan vaiheita ei toteutettu tiukasti peräkkäin. Etenkin vaiheet 3-5 olivat monilta osin samanaikaisia. Lisäksi vaiheita 3-5 toistettiin yhden syklin sisällä useina iteraatioina ketteriä kehitysmenetelmiä hyödyntäen ja tehden jatkuvaa arviointia, kuten Conboy, Gleasure ja Cullina (2015) sekä Sonnenberg ja Brocke (2012) suosittelivat.

Ensimmäisessä syklissä kehitettiin prototyyppinä työpöytäsovellus, jonka toimintojen pohjalta toisessa syklissä toteutettiin moderni web-sovellus. Ensimmäisessä syklissä etenkin vaihe 6 jätettiin vähälle huomiolle ja arviointi oli epämuodollisempaa. Toisaalta taas toisessa syklissä vaiheeseen 2 ei osoitettu yhtä paljon painoarvoa kuin ensimmäisessä, sillä suuri osa olennaisista tavoitteista oli jo aiemmin määritetty. Kunkin syklin voidaan vielä lisäksi ajatella jakautuvan rakentamis- ja arviointivaiheisiin, kuten March ja Smith (1995) kuvasivat. Vaiheet 1-3 kuuluvat rakentamisvaiheeseen ja vaiheet 4-6 arviointivaiheeseen.

4.3.1 Ensimmäinen sykli

Ensimmäinen vaihe käynnistyi omalla painollaan, kun yrityksessä havaittiin ongelma tietovarastointiprojektien toteuttamisessa. Tietyissä projekteissa oli olennaista olla käytettävissä sovellus tehostamaan kehitystä, mutta lähtötilanteessa yrityksellä ei ollut käytössä omaa sovellusta tähän tarkoitukseen. Eri asiakkuuksissa yritys oli käyttänyt useita erilaisia työkaluja tietovarastokehityksen apuna, mutta kaupallisten tuotteiden ongelma Suomen markkinoilla oli usein korkea hinta. Kehittämällä oman työkalun tietovarastojen kehitykseen yritys pyrki saamaan kilpailuetua ja parantamaan tarjontaa asiakkailleen. *Ongelmaksi* tunnistettiin omasta portfoliosta puuttuva tuote ja *motivaatioksi* kilpailuedun saaminen ja olemassaolevan tarjonnan parantaminen.

Vaiheessa kaksi pyrittiin määrittämään ratkaisulle tavoitteita. Koska tietovarastoprojektit Suomessa ovat yrityksen omaan kokemukseen perustuen maailman mittakaavassa usein varsin pieniä, on tarve kustannustehokkaalle vaihtoehdolle, kun kaupalliset tuotteet eivät mahdu projektien budjettiin. Toisaalta oli myös selkeästi tiedostettavissa, että saatavilla on jo työkaluja, jotka on suunniteltu toimimaan mahdollisimman monessa ympäristössä. Näin ollen nähtiin parhaaksi keskittää sovelluksen olennaisimmat logiikat yrityksen sisällä vallitseviin tapoihin toteuttaa tietovarastoratkaisuja. Omiin tarpeisiin räätälöidyllä työkalulla pyrittiin myös saamaan tietovarastotototeutuksista tasalaatuisia ja helposti ylläpidettäviä. Toisena päätavoitteena ohjelman kehityksen taustalla oli siis tarve tehostaa työn tekemistä sekä laatua. Näin ollen tavoitteiksi saatiin työn tehokkuuden ja laadun parantaminen. Työn tehokkuuteen liittyy olennaisena seikkana se, että sovelluksella voidaan toteuttaa koko tietovarastoratkaisu lähdelatauksista esityskerrokseen. Lisäksi sovelluksen tulee olla helppokäyttöinen, jotta sen avulla työskentely voi olla tehokasta, eli se on *käytettävä*.

Vaihe kolme aloitettiin valitsemalla toteutuksessa käytettävät teknologiat. Koska valtaosa potentiaalisista käyttökohteista oli Windows-ympäristössä. valittiin teknologiaksi Microsoftin .NET Framework ja siihen olennaisesti kuuluva C#-ohjelmointikieli. Sovelluksen data haluttiin tallentaa yhteen tietokantaan, minkä vuoksi tietokantamoottoriksi valittiin pilvessä toimiva Azure SQL Database. Tietokantaoperaatioiden toteuttamiseen valittiin .NET Frameworkiin hyvin sopiva Entity Framework. Teknologiavalintojen jälkeen määriteltiin sovellukseen kuuluvia olennaisimpia käsitteitä kuten tietokanta, tietokantataulu, näkymä, sarake

ja lataus. Näiden pohjalta työstettiin alustavat tietorakenteet ja operaatiot. Seuraavaksi sovellukseen lisättiin käyttöliittymä, joka toteutettiin WPF-kirjastolla (Windows Presentation Foundation), hyödyntäen Microsoftin Model-View-ViewModel -arkkitehtuuria. Käyttöliittymän kautta määritettiin olennaisia ominaisuuksia, joita tietovarastoratkaisun toteutuksessa tarvitaan. Kehitys tapahtui ketterällä prosessilla, jossa yhdessä loppukäyttäjien kanssa määritettiin jatkuvasti seuraavaksi toteutettavia toimintoja säännöllisten demonstraatioiden kautta. Demonstraatioissa loppukäyttäjillä oli mahdollisuus antaa palautetta toiminnoista. Palautteen perusteella toimintoja muokattiin ja uusia otettiin toteutettavaksi.

Arviointia tehtiin ensimmäisessä syklissä ennen toteutusta ja toteutuksen jälkeen. Vaiheisiin yksi ja kaksi liittyi olennaisesti sovelluksen toteutuksessa käytettävien teknologioiden valinta ennen toteutusta. Lisäksi motivaation sekä tavoitteiden määrittelyssä olennaisena osana oli sovelluksen hyödyllisyyden perustelu. Tämän vuoksi jouduttiin myös arvioimaan esimerkiksi sitä, kuinka sovellus maksaa itsensä takaisin. Vaiheissa yksi ja kaksi toteutettiin siis myös ex ante -arviointia. Demonstraatioihin liittyi edellä mainitun mukaisesti myös paljon jatkuvaa arviointia, mikä lopulta käynnisti toisen syklin.

4.3.2 Toinen sykli

Ensimmäisen syklin demonstraatioiden ja arvioinnin aikana havaittiin uusia tarpeita ja tavoitteita, joihin ei voitu vastata riittävän tehokkaasti kehityksen alla olevan työpöytäsovelluksen avulla. Näin ollen prosessin vaihe 1 tapahtui samanaikaisesti ensimmäisen syklin vaiheiden 4 ja 5 kanssa. Ensimmäisen syklin tuloksena syntynyt sovellus ei ollut riittävän käytettävä, jotta sen voitaisiin sanoa vastaavan aiemmin määriteltyä käytettävyyden vaatimusta. Asiakasympäristöön asennettava työpöytäsovellus ei tuntunut riittävän saavutettavalta, vaan koettiin tarve saada sovellus web-ympäristöön, jotta sen käyttäminen on helppoa mistä tahansa, eli sovellus on helposti saavutettava. Lisäksi aiemmin motivaatioksi tunnistettu kilpailuetu koettiin voitavan saavuttaa paremmin modernilla web-sovellksella. *Saavutettavuus* ja *moderni toteutus* tunnistetiin sovelluksen uusiksi tavoitteiksi, minkä vuoksi aloitettiin suunnittelemaan web-version toteutusta.

Toisen syklin vaihe 3 aloitettiin arvioimalla, mitä olemassaolevan prototyypin osia voidaan

hyödyntää web-version kehittämisessä. Sovelluksen tietorakenteet sekä niitä vasten suoritettavat operaatiot todettiin sopivan myös uuteen toteutukseen pääosin sellaisenaan. Sovellukselle tarvittiin uusi käyttöliittymä, jonka toteutus annettiin toisen henkilön vastuulle. Kirjoittajan vastuualueeksi toisessa syklissä määrittyi käyttöliittymän suunnittelu sekä sovelluksen *backend*, joka tämän toteutuksen kohdalla tarkoittaa käyttöliittymän kanssa keskustelevaa REST-rajapintaa sekä tietorakenteiden taustalla olevan tietokannan käsittelyä. Aiemmin käytetty .NET Framework todettiin huonosti sopivaksi web-toteutukseen, sillä se on sidottu Microsoftin teknologioihin koko sovelluksessa. Tästä syystä päätettiin siirtyä käyttämään alustariippumatonta .NET Core 2.2 -frameworkia. Tässä yhteydessä myös Entity Framework tuli vaihtaa EF Core -frameworkiin. .NET Core ja EF Core eivät tue kaikkia samoja ominaisuuksia kuin niiden vastaavat .NET Framework -komponentit, minkä vuoksi osia tietorakenteista ja tietokantaoperaatioista tuli refaktoroida uudelle alustalle. Uuden web-käyttöliittymän toteutukseen valittiin React.js -kirjasto. Lisäksi määritettiin tarkemmin sovelluksen ensimmäisen version vaatimat toiminnot, jotka on esitetty kuviossa 2.

Toisen syklin kehitys tapahtui samanlaisella ketterällä prosessilla kuin ensimmäisessä syklissä. Näin prosessivaiheita 3-5 suoritettiin useina iteraatioina, jotka sisälsivät säännöllisiä demonstraatioita loppukäyttäjille. Näin voitiin toteutettavien ominaisuuksien kehittämisessä ja suunnittelussa hyödyntää jatkuvasti loppukäyttäjien palautetta. Lisäksi näin toimien voitiin tehokkaasti priorisoida kulloinkin olennaisimpia ominaisuuksia toteutettaviksi.

Toteutusvaiheen jälkeen vuorossa oli laajempi arviointi tarkemmilla menetelmillä. Arviointi jaettiin eri osiin Pries-Hejen, Baskervillen ja Venablen esittämän viitekehyksen mukaan (ks. kuvio 1). Arviointia tehtiin teknisesti tarkastellen ohjelman toimintaa sekä laadullisesti loppukäyttäjille sekä liiketoiminnasta vastaaville henkilöille suunnatulla testauksella sekä kyselyllä. Loppukäyttäjien avulla arvioitiin ohjelman käytettävyyttä antamalla sovellus käyttäjien käytettäväksi, ja liiketoiminnan henkilöille esitettiin kysely. Loppukäyttäjien suorittamalla testauksella pyrittiin selvittämään, kuinka hyvin sovellus suoriutuu sille asetetuista laadullisista vaatimuksista, kuten käytettävyydestä ja työtehon parantamisesta. Liiketoiminnalle suunnattu kysely pyrki selvittämään, kuinka hyvin sovellus vastaa kilpailuedun ja tarjonnan parantamiseen. Teknisellä testaamisella pyrittiin saamaan tietoa esimerkiksi sovelluksen toimintavarmuudesta sekä tehokkuudesta. Lisäksi arvioitiin itse tutkimusprosessia tavoittee-

na selvittää, kuinka hyvin tutkimuksen toteutus on huomioinut suunnittelutieteellisen tutkimuksen ohjenuoria. Kaikki arviointi tässä vaiheessa kuului viitekehyksen ex post -osioon. Tekninen arviointi toteutettiin keinotekoisilla menetelmillä ja kohdistettiin artefaktiin. Kysely kuuluu luonnollisiin menetelemiin ja kohdistui artefaktiin. Tutkimuksen arviointi kuului luonnollisiin menetelmiin ja kohdistuu prosessiin. Taulukko 3 kuvaa arviointimenetelmien kohdistumisen vaatimuksiin. Arvioinnin tulokset esitellään luvussa 6.

Tämä tutkielma kattaa syklin viimeisen vaiheen, kommunikoinnin.

Taulukko 3: Arviointimenetelmien kohdistuminen vaatimuksiin

Menetelmä	Vaatimukset	Kohde	Menetelmän tyyppi
Kysely	Kilpailuetu, työn tehostaminen, laadun parantaminen	Artefakti	Luonnollinen
Beta-testaus	käytettävyys, työtehon parantaminen	Artefakti	Luonnollinen
Tekninen testaus	käytettävyys, työtehon parantaminen	Artefakti	Keinotekoinen
Prosessin arviointi	tutkimuksen laatu	Prosessi	Luonnollinen

Kuvio 2: Sovelluksen ensimmäiseen versioon määritellyt toiminnot.

5 Tutkimusartefakti

Tässä luvussa esitellään tarkemmin luvussa 4.3 esitetyn tutkimuksen tuloksena syntynyt artefakti. Tarkastelun kohteena on toisen kehityssyklin tuloksena syntynyt web-sovellus. Luvussa 5.1 kuvataan sovelluksen käyttötarkoituksia sekä ympäristöä. Luku 5.2 käsittelee sovelluksen arkkitehtuuria. Luvussa 5.3 käydään läpi sovelluksen toimintaa käyttöesimerkin kautta.

5.1 Sovelluksen käyttötarkoitukset ja ympäristö

Sovelluksella on kaksi päätarkoitusta:

- Tietovarastokehityksen tehostaminen
- Tietovarastoympäristön metadatan hallinta

Vaatimukset ovat vahvasti liitoksissa toisiinsa, sillä tietovarastokehityksen tehostamisessa hyödynnettävä automatisointi voidaan mahdollistaa tehokkaalla metadatan hallinnalla.

Kehitystä tehostetaan automatisoimalla kehitykseen liittyviä pakollisia mekaanisia toimintoja, jotka ilman vastaavaa työkalua ovat huomattavan työläitä. Tällaisia ovat esimerkiksi kenttien mappaamiset latauksissa, SQL-skriptien generointi sekä tietokantaobjektien luominen tietokantaan. Ohjelman avulla voidaan esimerkiksi luoda automaattisesti ajettava skripti, joka luo valitut objektit tietokantaan tarvittavine viite-eheysmäärityksineen sekä indekseineen.

Tietovarastoinnissa yksi olennaisimpia osia on tietomalli ja sen hyvä suunnittelu. Tietomallin suunnittelua helpottaa visuaalinen työkalu, jossa on vain olennaiset ominaisuudet: erilaisten taulutyyppien piirtäminen ja niiden välisten relaatioiden tekeminen helposti. Etenkin Data Vault -mallinnuksessa voidaan monesti jättää vähemmälle huomiolle relaatioiden kardinaalisuudet, sillä relaatiot ovat lähtökohtaisesti aina monesta moneen -tyyppisiä. Näistä syistä sovelluksessa on pyritty pitämään tietomallin piirtäminen mahdollisimman yksinkertaisena. Käyttäjä voi lisätä visuaaliseen malliin tauluja ja piirtää niiden välille relaatioita. Ohjelma määrittää automaattisesti vaadittavien viite-eheysmääritysten ja tarvittavien kenttien luomi-

sen.

Metadatan hallinta laajassa tietovarastossa on äärimmäisen tärkeää. Lähdejärjestelmiä voi olla tietovarastoon integroituna pienempien ympäristöjen muutamista suurien ympäristön kymmeniin. Mikäli tiedon lataamisessa käytettyä metadataa ei ole tallennettu mihinkään, voi tiedon alkuperäisen lähteen selvittäminen olla lähes mahdotonta. Metadatan tallennus tällaiseen työkaluun mahdollistaa data lineagen, eli tiedon kulkemisen lähteestä tietovarastoon, tarkastelun helposti mistä tahansa tietovarastokerroksesta eteen- tai taaksepäin. Näin saadaan aina tarvittaessa helposti selville, mistä data on alun perin ladattu. Metadatan tallennus siis myös samalla automaattisesti dokumentoi tietovarastoympäristön teknisiä yksityiskohtia.

Sovelluksessa metadata on olennaisessa osassa, sillä se toimii myös automaation mahdollistajana useissa eri tapauksissa. Kun tiedossa on taulujen kenttämääritykset tietotyyppeineen, voidaan erilaisten templaattien perusteella helposti generoida taulujen luontilauseita. Vastaavasti metadata mahdollistaa myös ETL-latauksissa tarvittavien SQL-skriptien generoinnin, jossa hyödynnetään samanlaisia templaatteja.

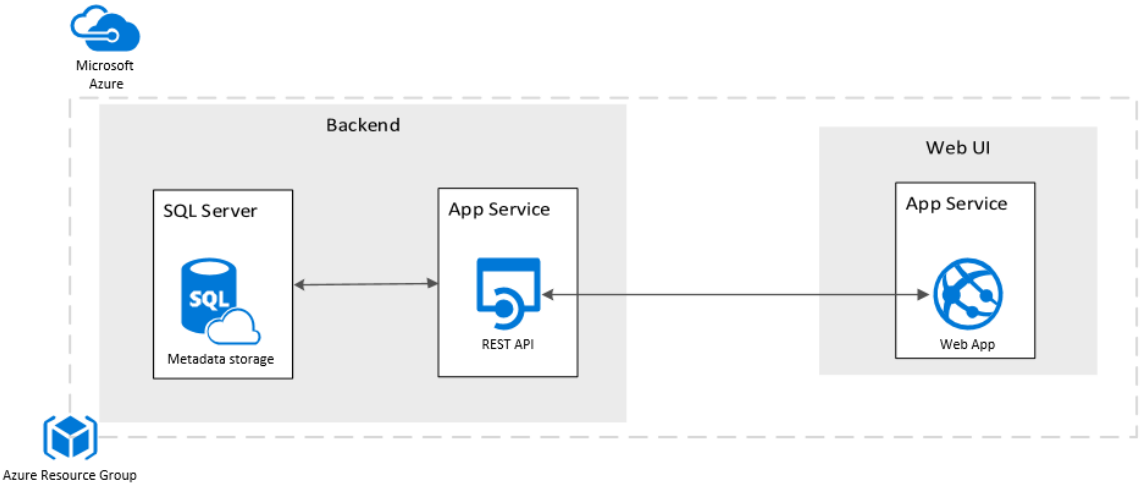
Yksittäisten latausten luomisen lisäksi sovelluksella voidaan määrittää useista latauksista koostuvia ajoketjuja. Ajoketjut liittyvät aina johonkin alustaan, ja käyttäjä voi luoda sovelluksessa määritetyn ajoketjun suoraan ajoalustalle. Ajoalustana voi toimia esimerkiksi Azure Data Factory tai Apache Airflow. Ajoketjun määritys on kuitenkin geneerinen, ja käyttäjä voi milloin vain vaihtaa ajoketjun toiselle alustalle, jolloin se voidaan generoida toiseen ympäristöön. Automaatiota hyödynnetään myös ajoketjujen generoinnissa; mikäli teknisistä syistä jokin taulu tulee ladata ennen toista taulua esimerkiksi viite-eheysvaatimusten vuoksi, käyttäjän ei tarvitse erikseen määrittää latausten suoritusjärjestystä.

Sovellus ja sen ominaisuudet on suunniteltu vahvasti organisaation omiin tarpeisiin. Näin ollen moni sen toiminnoista on suunniteltu ja toteutettu niin, että ne tukevat yrityksen sisällä vallitsevaa tapaa toteuttaa tietovarastoja. Tällaisiin päätöksiin kuuluvat esimerkiksi sovelluksella kehitetyn tietovarastoratkaisun arkkitehtuuri sekä tuetut teknologiavalinnat. Yrityksessä panostetaan vahvasti Data Vault -arkkitehtuuria noudattaviin ratkaisuihin ja uudet ratkaisut toteutetaan pääasiassa pilviympäristöihin. Näin ollen Data Vault -mallinnuksen sekä siihen

liittyvien latausprosessien automatisointi on ollut jatkuvasti etusijalla. Sovellus tukee tietovarastoalustana Snowflakea ja Azure SQL DB:tä, jotka ovat molemmat pilviympäristöissä toimivia relaatiotietokantoja. Koska sovellusta käytetään asiakasprojekteissa, sillä on luontaisesti vaikutuksia myös asiakasorganisaatioihin, vaikka käyttäjät ovat pääasiassa yrityksen omia työntekijöitä. Sovellus siis liittyy vahvasti sekä ympäristöönsä että siinä vallitseviin sosiaalisiin suhteisiin. Näin ollen sovellus voidaan nähdä teknologiapainotteista artefaktia laajempana järjestelmäartefaktina.

5.2 Arkkitehtuuri

Sovellus toimii Microsoftin Azure-ympäristössä. Pääkomponentit ovat käyttöliittymä ja backend. Backend koostuu Azure SQL DB -tietokannasta ja .NET Core 2.2 -frameworkilla toteutetusta REST-rajapinnasta. Kaikki komponentit kuuluvat samaan Azure Resource Groupiin. Kokonaisarkkitehtuuri on esitetty kuviossa 3.



Kuvio 3: Sovelluksen kokonaisarkkitehtuuri

Tarkemmalla tasolla backend koostuu useasta eri komponentista, jotka on kaikki toteutettu C#-ohjelmointikielellä .NET Core 2.2 -frameworkia hyödyntäen. Sovelluksen backend noudattaa pääosin Microsoftin (2019) määrittelemää ASP.NET Core MVC -arkkitehtuurimallia. Microsoftin MVC-mallissa (Model-View-Controller) sovellus jaetaan kolmeen komponenttien pääryhmään: *mallit*, *näkymät* ja *kontrollerit*. Tässä mallissa käyttäjäpyynnöt välitetään

kontrollerille. Kontrolleri suorittaa käyttäjäpyyntöjen mukaisia toimintoja malleja hyödyntäen. Kontrolleri valitsee käyttäjälle esitettävän näkymän ja välittää sille tarvittavat tiedot malleista. Tällaisella mallilla toteutettu sovellus pysyy Microsoftin mukaan hyvin skaalautuvana ja helposti testattavana.

5.2.1 API

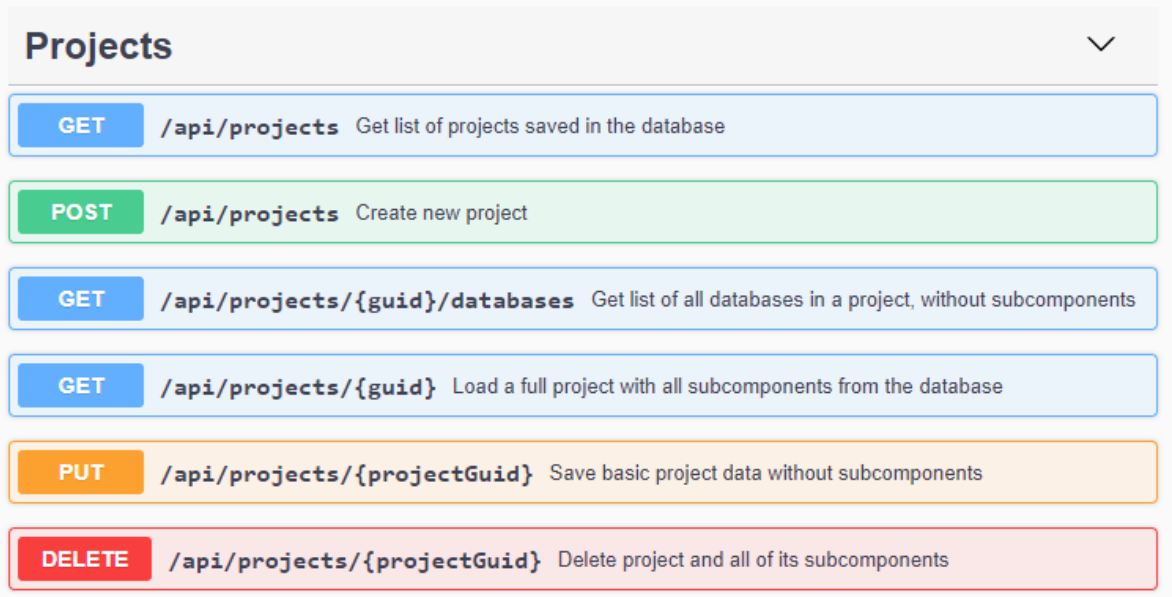
API-komponentti sisältää MVC-mallin kontrollerit. API on toteutettu REST (Representational state transfer) -arkkitehtuurimallia noudattaen, eli API on *RESTful*. Olennaisimpia suunnitteluperiaatteita tällaisen API:n toteuttamiseen Microsoftin (2018) ohjeistuksen mukaan ovat esimerkiksi seuraavat:

- API:n pääelementtejä ovat *resurssit*, eli mitkä tahansa objektit, tiedot tai palvelut joihin asiakasohjelmalla voidaan päästä käsiksi.
- Resurssilla on *tunniste*, jolla se voidaan uniikisti tunnistaa.
- Asiakkaat ovat vuorovaikutuksessa palvelun kanssa vaihtamalla resurssin *esitysmuotoja*, jotka voivat olla esimerkiksi JSON-määrityksiä.
- API on tilaton, eli HTTP-pyynnöt ovat itsenäisiä ja voivat tapahtua missä vaiheessa tahansa.

API on jaettu kontrollereihin sovelluksen pääresurssien perusteella, jotka ovat Connection, Database, Key, Mapping, Model, Project, Settings, Table, UserInfo, View ja Workflow. Jokainen kontrolleri sisältää vähintään sitä vastaavaan resurssiin liittyvät CRUD-operaatiot (Create, read, update and delete). Esimerkiksi ProjectsController -kontrolleri sisältää operaatiot projektien lisäämiseen, poistamiseen, muokkaamiseen ja hakemiseen. Esimerkki kontrollerin API-määrityksestä (ProjectsController) on esitetty kuviossa 4.

5.2.2 Tietokanta ja datan käsittely

Kontrollerit käsittelevät käyttäjien pyyntöjä hyödyntäen Models-komponentin luokkia. Models-komponentti sisältää MVC-mallin mallit, eli määritykset sovelluksen resurssien tietorakenteista sekä niiden operaatiot.



Kuvio 4: ProjectsController -kontrollerin API-määritys

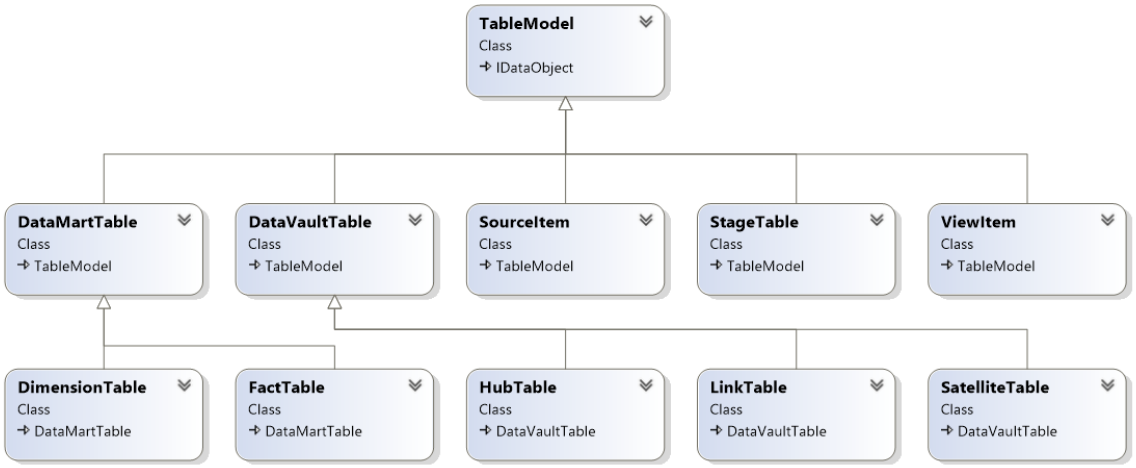
Kontrollerien ja mallien välissä on testattavuuden ja ylläpidettävyyden lisäksi erillinen abstraktiokerros, joka noudattaa Repository-mallia (Microsoft 2010). Repository-mallissa erotetaan datan käsittelylogiikka varsinaisesta sovelluslogiikasta, joka käyttää dataa. Tällöin sovelluslogiikka ei ole riippuvainen taustalla olevasta tiedon tallennusmenetelmästä. Tällöin esimerkiksi ProjectsControllerin (ks. kuvio 4) PUT-operaatio /api/projects/{projectGuid} välittää käyttöliittymältä tulevan datan repositoryn metodille SaveProject, joka hoitaa varsinaisen datan tallennuksen. Tällä erottelulla saavutetaan Microsoftin mukaan helpommin testattava ja ylläpidettävä sekä paremmmin muutoksiin mukautuva arkkitehtuuri.

Varsinaisena tiedon tallennuspaikkana sovelluksessa toimii pilvessä sijaitseva Azure SQL DB -relaatiotietokanta. Tietokannan käsittelyssä hyödynnetään Entity Framework (EF) Core -komponenttia. EF Core on kevyt, laajennettava avoimen lähdekoodin versio .NET Frameworkin Entity Framework -komponentista. Tällaista tiedonkäsittelykomponenttia käyttämällä voidaan tietokantaan tallentaa .NET -objekteja kirjoittamatta erikseen tietokannan käsittelyyn liittyvää SQL-koodia. (Microsoft 2016).

EF Coressa tietokannan käsittely tapahtuu mallin kautta, joka koostuu entiteettiluokista (Microsoft 2016). Olennaisena osana on integraatiotyökalu, jolloin tietokantaobjektien luominen ja muuttaminen voidaan tehdä pitkälle automatisoidusti suoraan sovelluksen tietomallin pe-

rusteella. EF Core siis suorittaa automaattisesti .NET objektin datan mappaamisen tietokantatauluihin. Datan käsittely tapahtuu EF Coren *tietokantakontekstin* avulla, joka esittää tietokantasessiota.Tietokannan kysely tapahtyy LINQ-lauseilla (Language Integrated Query). Microsoftin esimerkki tiedon hausta LINQ-lauseella:

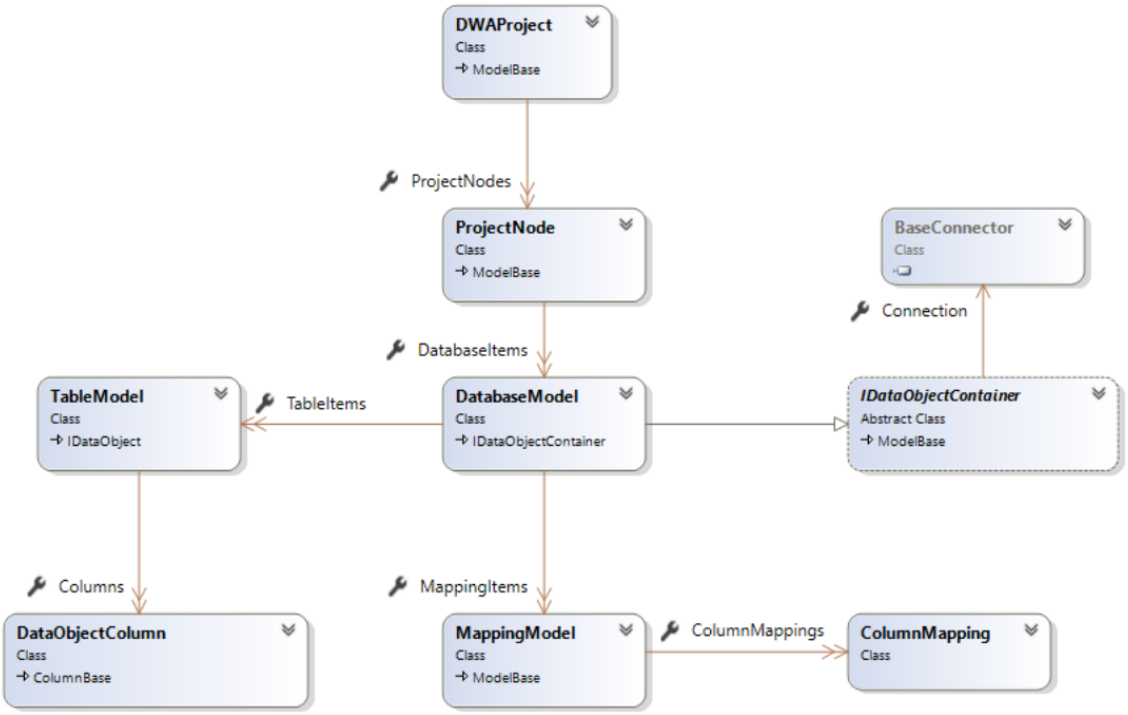
```
using (var db = new BloggingContext())
{
    var blogs = db.Blogs
        .Where(b => b.Rating > 3)
        .OrderBy(b => b.Url)
        .ToList();
}
```



Kuvio 5: TableModel-luokan periytyminen

Sovelluksen tietomalli noudattaa päätason rakenteeltaan samankaltaista rakennetta kuin kontrollerit. Tietomalli jakaantuu päätason objekteihin samaan tapaan kuin luvussa 5.2.1 kuvatut pääresurssit, jotka kuvaavat tietovarastoinnissa tarvittavia objekteja, vaikkakin jako kontrollereihin on tehty paljon itse tietomallia karkeammalla tasolla. Päätason luokista on peritty useita aliluokkia tarkempaan tyypitykseen. Esimerkiksi päätason TableModel-luokasta on peritty erilaisille taulutyypeille omat luokkansa, kuten StageTable, HubTable, LinkTable ja SatelliteTable. Kuvio 5 havainnollistaa geneerisen TableModel-luokan periytymisen tarkemmalle tasolle erilaisten tietokantataulutyyppien mukaan. Sovelluksen supistettu tietomalli,

joka kuvaa sovelluksen olennaisimmat tietorakenteet ja niiden suhteet, on esitetty kuviossa 6.



Kuvio 6: Sovelluksen supistettu tietomalli

5.3 Käyttöesimerkki

Tässä luvussa esitellään sovelluksen tarjoamia automatisointitoimintoja käyttöesimerkin kautta. Esimerkissä käydään läpi prosessi, joka sisältää seuraavat toimenpiteet:

- Lähdemääritys
- Latausmääritysten luominen lähteestä tietovarastoon
- Tietomallin määritys

Esimerkissä ovat voimassa seuraavat alkuvaatimukset:

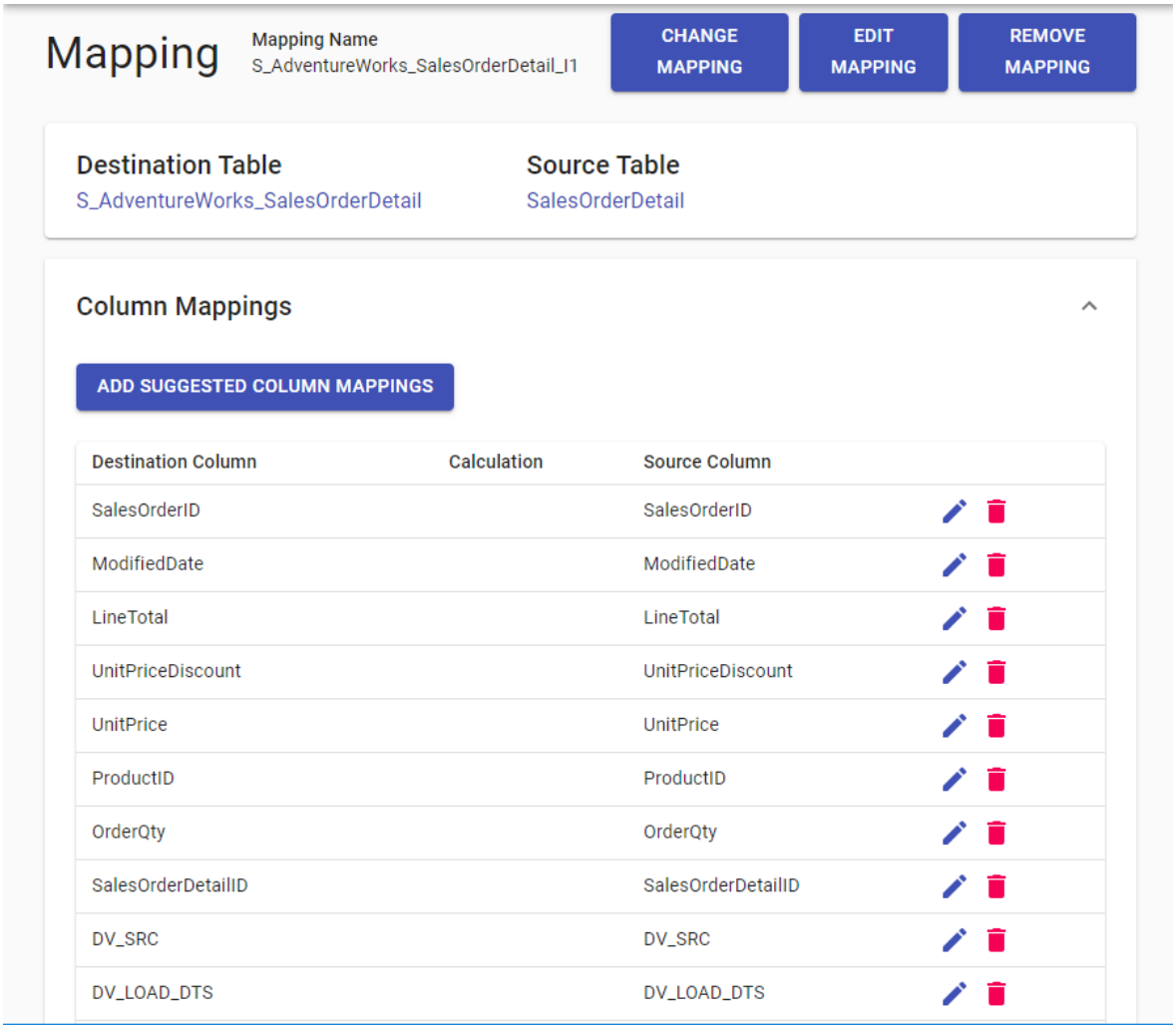
- Sovellukseen on määritetty tietokantayhteydet tarvittaviin tietokantoihin.
- Tiedon mallinnukseen tarvittavat avaimet ja datan lähde on tunnistettu.
- Sovellukseen on määritetty vaadittavat oletusarvot tiedoille, joita tarvitaan automati-

soinnissa, kuten oletustietotyypit sekä taulujen ja latausten nimeämiskäytännöt.

Esimerkissä ladataan dataa Microsoftin tarjoamasta AdventureWorks -tietokannasta. Lähdetauluksi on valittu tietokannan taulu dbo.SalesOrderDetail.

Prosessin ensimmäinen vaihe on tuottaa latausmääritykset datan saamiseksi lähdetietokannasta tietovarastoympäristöön, eli ns. staging-latauksen määritys. Käyttäjä valitsee sovelluksesta staging-tietokannan, johon data halutaan ladata. Staging-tietokannoilla on toiminto "Bring From Source", jolla latausmääritys saadaan tuotettua. Valittuaan toiminnon, käyttäjä valitsee ensin lähdetietokannan. Kun lähdetiedokanta on valittu, käyttöliittymä lähettää backendille pyynnön listata valitun tietokannan taulut. API tekee valittuun tietokantaan metadatakyselyn ja palauttaa käyttöliittymälle listan tietokannan sisältämistä tauluista. Käyttäjä valitsee lähdetaulun, jonka jälkeen käyttöliittymä lähettää backendille pyynnön listata valitun taulun sarakkeet. API tekee valittuun tietokantaan metadatakyselyn ja palauttaa käyttöliittymälle listan taulun sarakkeista. Käyttäjä valitsee tarvittavat sarakkeet. Seuraavaksi käyttäjä voi vielä määrittää uuden staging-taulun perusominaisuuksia, jotka on täydennetty projektille määritetyillä oletusasetuksilla. Esimerkkitapauksessa on määritetty oletuksena staging tauluille nimen etuliite "S_", ja varsinainen taulun nimi muodostetaan lähdetietokannan nimestä ja taulun nimestä. Tässä tapauksessa siis taulun koko nimeksi muodostuu S_AdventureWorks_SalesOrderDetail.

Kun käyttäjä on tarkistanut tiedot ja hyväksyy toiminnon suorittamisen, käyttöliittymä lähettää API:lle pyynnön luoda staging-latauksen vaatimat määritykset. Määritetyn metadatan pohjalta luodaan automaattisesti määritykset tietokantataululle ja sen rakenteelle sekä lataukselle. Oletetaan, että lähdejärjestelmä toimii eri tietokanta-alustalla kuin tietovarasto, jolloin tietotyypit eivät vastaa täysin toisiaan. Sovellukseen on määritetty kunkin alustan oletustietotyypit, jolloin staging-taulun tietotyyppimääritykset saadaan automaattisesti vastaamaan oikeaa alustaa lähdejärjestelmän tietotyyppien perusteella. Latausmääritys on saanut nimen automaattisesti kohdetaulun nimen ja siihen kohdistuvien latausten määrän mukaisesti, tässä tapauksessa S_AdventureWorks_SalesOrderDetail_I1. Lataukseen on automaattisesti määritetty, mikä lähdetaulun kenttä ladataan mihinkin kohdetaulun kenttään, sekä latausskriptissä mahdollisesti tarvittava tietotyyppimuutoksen logiikka. Esimerkki mapping-määrityksestä on esitetty kuviossa 7.

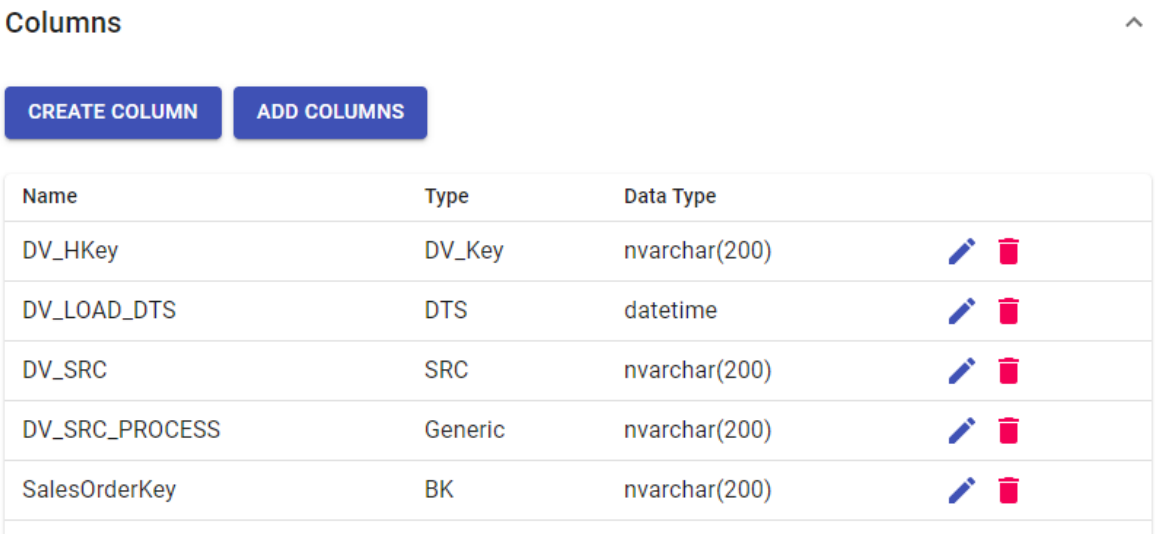


Kuvio 7: Esimerkki mapping-määrityksestä

Kun sovellus on luonut määrityksen staging-taulusta, voidaan sille määrittää varsinaisissa tietovarastolatauksissa käytettävät avaimet. Tarvittaviksi avaimiksi on tunnistettu SalesOrder ja SalesOrderDetail. Käyttäjä valitsee toiminnon "New Business Key", jolloin käyttöliittymä pyytää backendilta listan taulun sarakkeista, joista avain voidaan muodostaa ja esittää sen käyttäjälle. Käyttäjä muodostaa avaimen valitsemalla kentät, joista se koostuu. Tämän jälkeen käyttäjä voi vielä määrittää avaimen muodostamisessa tarvittavia parametreja, kuten ISNULL-arvon sekä kenttien erotinmerkin. Käyttöliittymä on täyttänyt automaattisesti projektille määritetyt oletusarvot. Tämän jälkeen käyttäjä painaa Build Calculation -painiketta, jolloin käyttöliittymä lähettää API:lle pyynnön muodostaa avaimelle määritettyjen parametrien perusteella sen vaatiman SQL-lauseen (calculation). Käyttäjä voi vielä muuttaa valintoja

ennen kuin hyväksyy muutokset.

Seuraavaksi siirrytään muodostamaan tietovaraston Data Vault -tietomalli. Tietomalli toteutetaan sovelluksessa visuaalisesti. Tietomalli koostuu kahdesta hubitaulusta, niiden satelliittitauluista sekä hubit yhdistävästä linkkitaulusta ja sen satelliitista. Käyttäjä antaa tauluille vain niiden nimen perusosan. Loput tarvittavat tiedot syntyvät automaattisesti etukäteen määritettyjen sääntöjen mukaan. Käyttäjä valitsee toiminnon Create Table, jolloin hänelle aukeaa dialogi, jossa taulun ominaisuudet määritetään. Käyttäjä luo ensin hubitaulun SalesOrder, joka saa esimerkkimäärityksistä kokonaisnimen RV_SalesOrder_H. Kun käyttäjä hyväksyy taulun perusominaisuudet, käyttöliittymä pyytää backendia luomaan taulun määrityksen. Backend luo taulun ja sille oletusasetuksissa määritetyt kentät, ja palauttaa käyttöliittymälle uuden taulun tiedot. Esimerkki hubitaululle luoduista oletuskentistä on esitetty kuviossa 8. Seuraavaksi käyttäjä luo vastaavasti toisen hubitaulun RV_SalesOrderDetail_H.



Kuvio 8: Hubitaulun automaattisesti luodut kenttämääritykset

Tämän jälkeen käyttäjä piirtää yhteyden kahden hubitaulun välille, jolloin käyttöliittymä esittää käyttäjälle esitäytetyn dialogin linkkitaulun luomiseen. Linkkitaulu saa hubien nimien perusteella automaattisesti nimiehdotuksen RV_SalesOrder_SalesOrderDetail_L. Käyttäjä voi myös valita automaattisesti tehtäväksi linkille validisuussatelliitin, jolloin samalla syntyy myös taulu RV_SalesOrder_SalesOrderDetail_LVS. Kun käyttäjä hyväksyy linkkitaulun perusominaisuudet, käyttöliittymä lähettää backendille pyynnön luoda linkkitaulu ja sille vas-

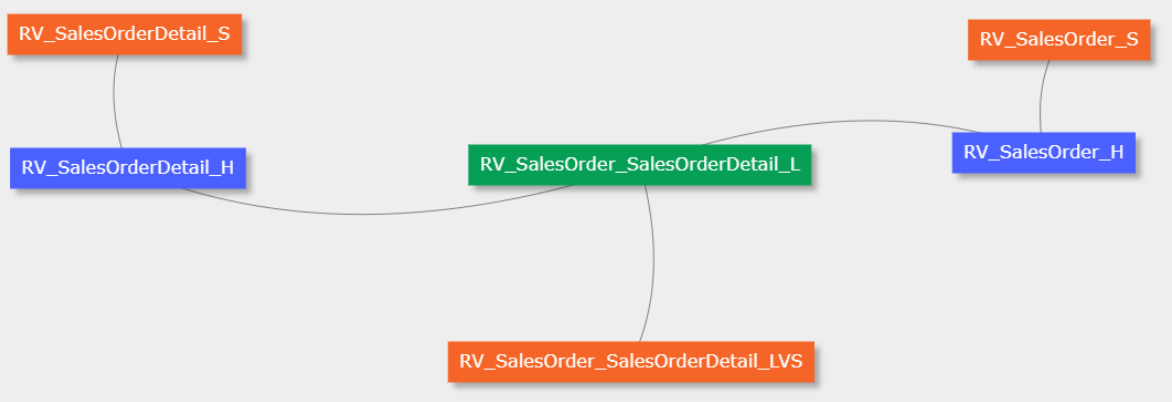
taava validisuussatelliitti, ja palauttaa käyttöliittymälle uusien taulujen tiedot. Linkkitaululle luodaan automaattisesti oletuskentät, sekä lisäksi viiteavainkenttä jokaista siihen liittyvää hubia kohden. Lopputuloksena on linkkitaulu, josta on yhteys edellä mainittuihin kahteen hubitauluun sekä uuteen satelliittitauluun.

Käyttäjä lisää vielä hubeille satelliittitaulut valitsemalla ensin hubitaulun ja sitten toiminnon "Create Satellite". Satelliittitaulu saa nimiehdotuksen valittuna olevan hubitaulun perusteella, tässä tapauksessa RV_SalesOrderDetail_S sekä RV_SalesOrder_S. Kun käyttäjä hyväksyy satelliittitaulun perusominaisuudet, käyttöliittymä lähettää backendille pyynnön luoda satelliittitaulu. Backend luo uuden satelliittitaulun, sille oletuskentät ja palauttaa uuden taulun tiedot käyttöliittymälle. Lopputuloksena on satelliittitaulu, josta on yhteys hubitauluun.

Jokaiselle taululle käyttäjä voi määrittää automaattisesti mapping-määritykset valitsemalla New Input Mapping -toiminnon. Tällöin käyttöliittymä esittää käyttäjälle listan lähdetauluista. Kun käyttäjä on valinnut lähdetaulun ja hyväksyy mapping-määrityksen perusominaisuudet, käyttöliittymä lähettää backendille pyynnön luoda uusi mapping. Tämän jälkeen käyttäjälle esitetään mahdollisuus siirtyä määrittämään tarkemmin kenttämappingit edellä luodulle lataukselle. Käyttäjä valitsee Add Suggested Mappings -toiminnon, jolloin käyttöliittymä pyytää backendilta listan sopivista kenttämappauksista. Käyttäjä valitsee listalta halutut mappaukset, ja hyväksymisen jälkeen käyttöliittymä lähettää backendille pyynnön lisätä kenttämappaukset mappingin tietoihin.

Satelliittitaulut ovat edellä luoduista tauluista ainoita, jotka tarvitsevat lisää kenttiä sovelluksen automaattisesti luomien lisäksi. Käyttäjä valitsee Add Columns -toiminnon, jolloin käyttöliittymä pyytää backendilta listan sovellukseen määritetyistä staging-tauluista ja esittää sen käyttäjälle. Käyttäjä valitsee sopivan lähdetaulun ja taulusta haluamansa kentät. Kun halutut kentät on määritetty, käyttöliittymä lähettää backendille pyynnön lisätä kentät tauluun. Backend lisää taululle halutut kentät sekä luo mapping-määrityksen, joka sisältää kenttämappaukset edellä valituista lähdetaulun kentistä uusiin satelliitin kenttiin. Mikäli mapping on jo olemassa, uuden kenttämappaukset lisätään olemassaolevaan mapping-määritykseen. Lopullinen tietomalli on esitetty kuviossa 9.

Edellä kuvatussa esimerkissä määritettiin metadata tietovaraston tietomallille ja sen popu-



Kuvio 9: Sovelluksessa toteutettu Data Vault -tietomalli

lointiin tarvittaville latausmäärityksille. Kun metadata on tallennettu sovellukseen, voidaan varsinaiset objektit generoida sen pohjalta helposti hyödyntämällä templaatteja. Esimerkiksi SQL Server -ympäristöissä satelliittitaulun luontilauseen templaatti on määritetty seuraavasti:

```
-- @dwa_table_full_name
USE [@dwa_database]

IF NOT EXISTS (SELECT * FROM [sys].[objects]
WHERE [object_id] = OBJECT_ID(N'@dwa_table_with_schema') AND
[type] in (N'U'))
CREATE TABLE [@dwa_table_schema].[@dwa_table_full_name]
(
@dwa_columns,
@dwa_constraint
)
```

Kun templaatti täydennetään esimerkissä luodun RV_SalesOrderDetail_S -satelliittitaulun metadatalla, saadaan helposti generoitua taulun ajettava luontilause:

```
-- RV_SalesOrderDetail_S
USE [solteqdwa-dw]
```

```
IF NOT EXISTS (SELECT * FROM [sys].[objects]
  WHERE [object_id] = OBJECT_ID(N'dbo.RV_SalesOrderDetail_S')
  AND [type] in (N'U'))
CREATE TABLE [dbo].[RV_SalesOrderDetail_S]
(
  DV_HKey nvarchar(200) NOT NULL,
  DV_LOAD_DTS datetime NOT NULL,
  DV_SRC nvarchar(200) NOT NULL,
  DV_SRC_PROCESS nvarchar(200) NOT NULL,
  SalesOrderID int ,
  ModifiedDate datetime ,
  LineTotal numeric(12,4) ,
  UnitPriceDiscount numeric(12,4) ,
  UnitPrice numeric(12,4) ,
  ProductID int ,
  OrderQty smallint ,
  SalesOrderDetailID int ,
  rowguid uniqueidentifier ,
  CONSTRAINT RV_SalesOrderDetail_S_PK PRIMARY KEY(DV_HKey),
  CONSTRAINT RV_SalesOrderDetail_S_FK FOREIGN KEY(DV_HKey)
  REFERENCES dbo.RV_SalesOrderDetail_H(DV_HKey)
);
```

Vastaavasti esimerkissä luodun hubitaulun RV_SalesOrder_H -populoinnissa tarvittavan SQL-skriptin templaatti on määritetty seuraavasti:

```
INSERT INTO @dwa_table_schema@.@dwa_table_full_name@
(
  @dwa_columns@
)
(
  SELECT DISTINCT
```

```
  @dwa_hub@ as @dwa_hub_dvkey@,
  @dwa_source_dts@ AS @dwa_destination_dts@,
  @dwa_source_src@ AS @dwa_destination_src@,
  '@dwa_mapping_name@' AS DV_SRC_PROCESS,
  src.@dwa_hub_source_key@ AS @dwa_hub_key@
  FROM @dwa_source_schema@.@dwa_source_table@ src
  -- Target table check
  LEFT JOIN @dwa_table_schema@.@dwa_table_full_name@ trg
  ON src.@dwa_hub_source_key@ = trg.@dwa_hub_key@
  -- Insert only new rows
  WHERE trg.@dwa_hub_key@ IS NULL
);
```

Kun edellä esitetty templaatti populoidaan latausmäärityksen RV_SalesOrder_H_I1 -metadatalla, saadaan ajettava skripti:

```
INSERT INTO VAULTDATA.RV_SalesOrder_H
(
  DV_HKey,
  DV_LOAD_DTS,
  DV_SRC,
  DV_SRC_PROCESS,
  SalesOrderKey
)
(
  SELECT DISTINCT
  SalesOrderHashKey as DV_HKey,
  src.DV_LOAD_DTS AS DV_LOAD_DTS,
  src.DV_SRC AS DV_SRC,
  'RV_SalesOrder_H_I1' AS DV_SRC_PROCESS,
  src.SalesOrderKey AS SalesOrderKey
  FROM STG.S_AdventureWorks_SalesOrderDetail src
```

```
  -- Target table check
  LEFT JOIN VAULTDATA.RV_SalesOrder_H trg
  ON src.SalesOrderKey = trg.SalesOrderKey
  -- Insert only new rows
  WHERE trg.SalesOrderKey IS NULL

);
```

Metadataa hyödyntäen käyttäjä säästyy kirjoittamasta käsin toisteisia, mahdollisesti hyvinkin pitkiä SQL-skriptejä, joita tietovaraston tietomalli sekä latausmääritykset vaativat. Lisäksi tallennettu metadata mahdollistaa helposti jälkikäteen tietovirtojen selvityksen. Mikäli halutaan tietää, mistä lähteestä jonkin yksittäisen satelliittitaulun kentän data on peräisin, sovelluksen avulla tämä tieto on helposti näytettävissä. Metadata mahdollistaa myös helposti toisiinsa liittyvien objektien generoinnin, sillä metadatan perusteella voidaan päätellä, mitkä entiteetit liittyvät toisiinsa. Mikäli halutaan kerralla generoida tietovarastoympäristöön jonkin tietovarastotaulun lataamiseen liittyvät objektit, ne voidaan metadatan perusteella generoida muutamalla painalluksella. Esimerkiksi linkkitauluun liittyy aina vähintään kaksi viittausta hubitauluihin, mahdollisesti validisuussatelliitti sekä näihin tauluihin liittyvät latausmääritykset. Lisäksi metadatan avulla luodaan objektit automaattisesti oikeassa järjestyksessä, mikäli esimerkiksi viite-eheysmääritysten vuoksi yhtä taulua ei voida luoda ennen kuin toinen siihen liittyvä taulu on jo olemassa.

Ilman metadataa ja automaattista generointia kaikki objektit joudutaan luomaan käsin. Kehittäjän aikaa kuluu manuaaliseen työhön, joka edellä esitetyn mukaisesti on monilta osin automatisoitavissa.

6 Arviointi

Tässä luvussa arvioidaan tutkimusta ja sen tuloksia luvussa 4 esitettyjen menetelmien mukaisesti (ks. taulukko 3 menetelmien kohdistumisesta vaatimuksiin). Luvussa 6.1 arvioidaan, kuinka hyvin sovellus vastaa taustalla olleisiin liiketoiminnallisiin tarpeisiin. Luvussa 6.2 arvioidaan sovelluksen käytettävyyttä. Luvussa 6.3 arvioidaan sovelluksen teknistä suorituskykyä ja lopuksi luvussa 6.4 arvioidaan tutkimusprosessin onnistumista.

6.1 Kysely liiketoiminnan edustajille

Koska sovelluksen kehittämisen taustalla olivat liiketoiminnalliset tarpeet, oli tarpeellista arvioida, kuinka sovellus vastaa liiketoiminnan tarpeisiin. Arviointia varten esitettiin lyhyt kysely yrityksen Business Intelligence -liiketoiminnasta vastaavalle henkilölle sekä myyntiorganisaation edustajalle. Sovelluksen kehityksen taustalla olleita liiketoiminnallisia ongelmia käsiteltiin tarkemmin luvussa 4.3.1. Kysely sisälsi seuraavat kysymykset:

1. Tietovarastokehityksen automatisointisovelluksen taustalla oli omasta portfoliosta puuttuva tuote. Miksi tällaisen sovelluksen kehittäminen oli yritykselle tärkeää?
2. Sovelluksen kehityksen motivaationa oli kilpailuedun saaminen ja yrityksen tarjonnan parantaminen. Kuinka näet kehitetyn sovelluksen vastaavan näitä tavoitteita?
3. Kuinka yrityksen itse omiin tarpeisiin kehittämä automatisointisovellus mielestäsi tukee tavoitteeksi asetettua työn tehokkuuden ja laadun parantamista?
4. Millaisia hyötyjä kehitetty sovellus tarjoaa liiketoiminnan näkökulmasta?

Ensimmäisen kysymyksen vastauksissa korostettiin automatisointisovelluksen tarvetta sillä, että Data Vault -tietovarastot ovat jatkuvasti suuremmassa roolissa yritystä eniten kiinnostavien asiakkaiden keskuudessa. Päätöksessä siis vaikuttivat vahvasti yrityksen asiakkaiden tarpeet. Yrityksessä kartoitettiin markkinoilta sopivia tuotteita, mutta esimerkiksi puutteellisen toiminnallisuuden tai korkean hinnan vuoksi sopivaa ei löytynyt. Kuitenkin oli selkeää, että automatisointisovellus tarvitaan, joten se päädyttiin rakentamaan itse.

Toisessa kysymyksesä arvioitiin kilpailuedun saamista ja tarjonnan parantamista. Vastauk-

sista nousi ilmi se, että sovellus ei ole vielä kovin laajasti käytössä, joten näihin seikkoihin ei vielä voida vastata täsmällisesti. Toisaalta ilmeni, että vaikka sovellus on vielä kehitysvaiheessa, se on esimerkiksi avannut mahdollisuuksia osallistua tarjouskilpailuihin, joihin ilman vastaavaa sovellusta ei olisi päästy mukaan. Lisäksi vastaanotto asiakkaiden suunnalta on ollut positiivista.

Kolmannessa kysymyksessä arvioitiin työn tehokkuuden ja laadun parantamista. Vastauksissa tuli ilmi se, että automatisointi parantaa etenkin ratkaisujen ylläpidettävyyttä, sillä yksittäisen kehittäjän kädenjälki näkyy ilman vastaavaa sovellusta tehdyissä projekteissa usein liian paljon. Lisäksi esille nousi asiakkaiden suunnalta tulevat vaatimukset, joiden vuoksi yrityksen tarjonnassa tarvitaan automatisointisovellus. Työn tehokkuuden paraneminen ja tasalaatuiset, ylläpidettävät toteutukset todettiin olevan myös asiakkaiden etu.

Neljännessä kysymyksessä arvioitiin sovelluksen tarjoamia liiketoiminnallisia hyötyjä. Kuten edellä toisessa kysymyksessä, myös tämän kysymyksen vastauksien perusteella automatisointisovellus tuo yritykselle selkeää kilpailuetua ja se on tärkeä osa yrityksen tulevaisuutta. Sovelluksen avulla yritys voi tarjota asiakkailleen entistä nopeamman ja ketterämmän liikkeellelähdön datan hyödyntämisessä.

6.2 Käytettävyyden testaus

Koska sovelluksen vaatimukseksi oli asetettu käytettävyys ja motivaationa oli työskentelyn tehostaminen, haluttiin sovellusta arvioida näitä kriteerejä vasten. Arviointi tehtiin betatestauksena, jossa sovellus annettiin loppukäyttäjien testattavaksi ja pyydettiin heiltä kommentteja sovelluksen toimivuudesta, pääpainona sujuva työnkulku. Arvointimenetelmän valinnassa vaikuttivat ajankäytöstä ja kustannuksista seuraavat rajoitteet. Esimerkiksi Bruun ym. 2009 vertailivat perinteistä ohjattua käytettävyystestausta etänä suoritettaviin menetelmiin, jossa käyttäjät raportoivat ongelmat omatoimisesti. Vaikka tehokkuus ei ollut ohjatun käytettävyystestauksen tasolla, omatoimisesti raportoitavat käytettävyystestit olivat merkittävästi kustannustehokkaampia. Smilowitzin, Darnellin ja Bensonin (1994) arvioinnissa betatestauksella saavutettiin lähes yhtä hyviä tuloksia kuin ohjatulla käytettävyystestauksella. Lisäksi loppukäyttäjien tiedettiin olevan kokeneita tietovarastokehittäjiä, jotka eivät tarvitse

ohjausta sovelluksen tarjoamien toimitojen suorittamiseen.

Käytettävyystestauksessa sovelluksen käyttöliittymää pidettiin pääosin selkeänä ja käyttäjillä ei ilmennyt suuria ongelmia toimintojen suorittamisessa. Huomiota kiinnitettiin erityisesti navigaatioon ja usein toistuviin toimintoihin, sillä pääpaino testauksessa oli selkeän työnkulun ja työn tehostamisen testaamisessa. Kun käyttäjät olivat kirjanneet huomioita sovelluksen käytettävyydestä, huomiot käytiin läpi yhdessä kehittäjien ja käyttäjien kanssa, jotta voitiin valita tarkasteltavaksi ja korjattaviksi olennaisimmat epäkohdat.

Sovelluksessa navigoinnissa havaittiin joitain ongelmia, joiden korjaamista pidetiin tärkeänä. Sovellus on rakennettu niin, että se mahdollistaa tietovarastoympäristöön mielivaltaisen määrän tietokantoja eri tasoille (staging, data vault, information mart). Näin ollen navigoinnissa tarjottiin aina jotain tasoa valitessa lista tietokannoista. Todettiin, että useimmissa tapauksissa tietokantoja on vain yksi jokaista tasoa kohden. Näin ollen päädyttiin ratkaisuun, jossa tietokantalista esitetään vain ensimmäisen kerran, kun käyttäjä valitsee jonkin tason. Kun käyttäjä on kertaalleen valinnut jonkin tason alta tietokannan, seuraavan kerran käyttäjän valitessa saman tason esitetään suoraan aiemmin valitun tietokannan taululistaus. Samalla poistettiin tietokantatason navigaatiopalkki, ja tietokannan vaihtaminen päädyttiin siirtämään sivuun valikon alle, jolloin muulle sisällölle vapautui tilaa.

Taulun yksityiskohtaisessa näkymässä oli aluksi painikkeet taulun perusominaisuuksien muokkaamiseen, taulun vaihtamiseen ja taulun poistamiseen. Nämä painikkeet koettiin yksityiskohtaisessa näkymässä turhina, ja toiminnot siirrettiin taululistaukseen, jotta yksityiskohtaisessa näkymässä saatiin vapautettua enemmän tilaa muulle sisällölle. Vaikka itse toimintojen suorittamisessa ei ollut suuria ongelmia, osassa toimintopainikkeista oli käyttäjille epäselvää, minkä toiminnon painike laukaisee. Tämän seurauksena toimintopainikkeiden tekstejä yhdenmukaistettiin, jotta käyttäjä löytää tarvitsemansa toiminnot helposti.

Olennaisena työnkulkua hankaloittavana tekijänä koettiin lähdelatauksiin liittyvien käsitteiden erottelu erillisiin lähdelatauksiin, mappingeihin ja kohdetauluihin. Koska lähdelatauksessa ei välttämättä käsitellä suoraan yksittäistä taulua, todettiin, että lähdelataukseen liittyvän SQL-kyselyn määrityksen tulee olla kiinteämpi osa mapping-määritystä. Näin toimien työnkulku koettiin saatavan sujuvammaksi, kun loogisesti samaan asiaan liittyviä osia saatiin

yhdistettyä.

Alkutilanteessa sovelluksen eri tason navigaatiopalkit olivat saman värisiä. Navigaatiopalkkien väritystä pyrittiin säätämään, jotta eri navigaatiotasot näkyisivät käyttäjälle selkeämmin.

Käyttäjätestauksella saatiin esille sovelluksen piirteitä, jotka voivat hankaloittaa sen käyttöä. Edellä mainittujen lisäksi käyttäjätestauksessa tuli ilmi myös muita mahdollisia ongelmakohtia, joita ei kuitenkaan koettu kriittisiksi. Testauksessa valittiin välittömästi korjattaviksi olennaisimmat ongelmat, ja loput kirjattiin ylös odottamaan jatkokehitystä.

6.3 Tekninen testaus

Koska oli tiedossa, että tietovarastoympäristöt voivat olla hyvin laajoja, teknisessä testauksessa keskityttiin tarkastelemaan sovelluksen suorituskykyä suurien objektimäärien kanssa. Sovellukseen luotiin tavanomaista ympäristöä suurempi määrä tauluja sekä mapping-määrityksiä. Määrät valittiin yrityksen aiempien tietovarastoprojektien perusteella sellaisiksi, että objekteja on huomattavasti enemmän kuin keskimäärin toteutetuissa projekteissa. Tämän jälkeen tarkistettiin sovelluksen rakenteissa navigoimisessa tarvittavien ydintoimintojen kestoja. Testattaviksi valitut määrät:

- 1000 staging-taulua, joista jokaisella 20 saraketta, 5 business keytä ja 1 mapping-määritys
- Jokaista staging-taulua kohden 2 hubitaulua, yksi linkkitaulu ja yksi satelliittitaulu (yhteensä 4000 Data Vault -taulua), lisäksi mapping-määritys jokaiseen tauluun. Jokaisella hubitaululla 5 kenttää, jokaisella linkkitaululla 5 kenttää ja jokaisella satelliittitaululla 25 kenttää

Testattavat ydintoiminnot:

- Haetaan tietokannan taulut
- Haetaan tietokannan mapping-määritykset

Toimintojen kestojen arvioinnissa hyödynnettiin kirjallisuudessa esitettyjä arvoja. Nielsen (1993) tuo esille 3 raja-arvoa toimintojen vasteajoille: 0.1 sekuntia, 1 sekunti ja 10 sekun-

tia. 0.1 sekunnin kestävä toiminto koetaan välittömänä, 1 sekunnin viiveen kohdalla käyttäjän huomio voi herpaantua, ja yli 10 sekunnin kesto on liikaa pitämään käyttäjän huomion kiinnittyneenä toiminnon suorittamiseen. Palmerin 2002 mukaan web-sivujen tulisi latautua "muutaman sekunnin sisällä". Käyttäjien kärsivällisyyden Palmer toteaa alkavan kärsiä noin 8 sekunnin kohdalla ja 10 sekunnin jälkeen on todennäköistä, että käyttäjä ei odota sivun latautumista.

Tietokannan taulujen hakua testattiin hakemalla testitapaukseen luodut 4000 Data Vault -taulua. Haku tehtiin useita kertoja, ja keskimääräiseksi hakuajaksi saatiin 19.4 sekuntia. Käyttöliittymälle palautetun datan koko oli 29 megatavua. Tässä tapauksessa haun kesto todettiin liian pitkäksi ja huomattiin, että taulujen haun yhteydessä haetaan osittain turhaa tietoa. Tauluja listatessa ei ole esimerkiksi vielä tarpeen hakea tietoa taulujen sarakkeista. Taulujen hakua muutettiin niin, että se palauttaa taulusta vain perustiedot, jotka tarvitaan taululistauksen näyttämiseen käyttäjälle. Muutoksen jälkeen haun testaus toistettiin, ja keskimääräiseksi hakuajaksi saatiin 5.23 sekuntia ja palautetun datan kooksi 2.86 megatavua. Tämän todettiin olevan edellä mainittujen raja-arvojen sisällä, ja siten hyväksyttävä aika.

Tietokannan mappingien hakua testattiin hakemalla testitapaukseen luodut 4000 mappingia. Haku tehtiin useita kertoja ja keskimääräiseksi hakuajaksi saatiin 5.45 sekuntia. Käyttöliittymälle palautetun datan koko oli 902.71 kilotavua. Myös tämä todettiin olevan edellä mainittujen raja-arvojen sisällä ja siten hyväksyttävä aika.

Sovelluksessa navigoinnin ohella haluttiin arvioida joitain toimintoja yksittäisen, merkittävästi tavallista suuremman objektin kanssa. Tätä varten luotiin keinotekoinen lähdetaulu, jossa on 1000 kenttää. Tästä taulusta tehtiin automaattisen Bring from source -toiminnon avulla staging-taulu ja lähdemappingin määritys. Toiminnon keskimääräiseksi kestoksi saatiin 29.3 sekuntia ja käyttöliittymälle palautetun datan kooksi 1.11 megatavua. Kestoa pidettiin pitkänä, ja tarkemman analyysin perusteella > 90% käytetystä ajasta kului tietokantaoperaatioihin. Testiympäristössä oli käytössä tietokantana minimitehoille asetettu Azure SQL DB -tietokanta (Basic 5 DTU). Tuotantoympäristön simuloimiseksi tietokanta skaalattiin tehokkaammaksi (Standard 20 DTU) ja suoritettiin sama toiminto uudelleen. Uudelleenskaalauksen jälkeen toiminnon kesto lähes puolittui ollen keskimäärin 15.4 sekuntia. Koska kesto ylitti silti 10 sekunnin raja-arvon, tehtiin testaus uudelleen pienemmällä taululla, jossa oli

300 kenttää. Tällöin kestoksi saatiin keskimäärin 6.1 sekuntia, mikä oli raja-arvojen sisällä ja testitapauksen kenttien määrä tavanomaisen skaalan yläpäässä. Huomioiden ensimmäisen testitapauksen poikkeuksellisen suuren koon ja tietokanta-alustan tehojen vaikutuksen, todettiin toiminnon suorittaminen riittävän nopeaksi. Havaittiin kuitenkin potentiaalinen käytettävyysongelma, joka voi vaatia toimenpiteitä myöhemmin.

Seuraavaksi luotiin tyhjä satelliittitaulu, jolle lisättiin staging-taululle edellä määritetyt 1000 kenttää ja mapping-määritys automaattisen Add columns -toiminnon avulla. Tässä vaiheessa käytettiin samaa ylösskaalattua tietokanta-asetusta kuin edellä mainittu. Toiminnon kestoksi saatiin keskimäärin 7.8 sekuntia ja käyttöliittymälle palautettava datamäärä oli 739 kilotavua. Kuten edellä, valtaosa suoritusajasta kului tietokantaoperaatioihin. Koska kesto oli lähellä raja-arvojen ylärajoja, testattiin toiminto vielä edellä luodulla 300:n kentän taululla, jolloin kesto oli keskimäärin 1.9 sekuntia. Toiminto todettiin riittävän nopeaksi samoista syistä kuin edellä käsitelty Bring from source -toiminto.

Koska mallinnus on sovelluksessa olennainen toiminto, haluttiin testata vielä mallinnusta suurella tietomallilla. Mallinnuksen lähtökohta sovelluksessa on, että pääasiassa käsitellään pieniä tietomalleja, jotka käsittävät jonkin yksittäisen kohdealueen. Tämän lähtökohdan perusteella testitapaukseksi otettiin 200:n taulun tietomalli, joka on merkittävästi suurempi kuin keskimääräinen tarkasteltava tietomallin osa. 200:n taulun tietomallin lataaminen oli nopeaa ja uusien objektien lisääminen taululle onnistui sujuvasti. Mallin käsittely onnistui sujuvasti, vaikka objektien siirto paikasta toiseen ei ollut koko ajan yhtä sujuvaa kuin pienemmän mallin kanssa, ja viive pysyi alle esitetyn 0.1 sekunnin raja-arvon.

Suorituskykytestien perusteella sovellus toimii ympäristössään suurillakin objektimäärillä riittävän hyvin, jotta sen käytettävyys ei kärsi. Ainoa selkeä toimenpiteitä vaatinut suorituskykyongelma taulujen listauksessa saatiin korjattua niin, että suorituskyky saatiin riittävälle tasolle. Mallien käsittelyssä havaittiin mahdollinen ongelma suorituskyvyssä, mikäli malli kasvaa hyvin suureksi. Toisaalta on tietoinen suunnittelupäätös, että sovelluksella on tarkoitus käsitellä vain pieniä malleja kerrallaan, eikä se sovellu hyvin koko tietovarastoympäristön mallin tarkasteluun. Näin ollen visuaalisen tietomallin osalta ei tällä hetkellä nähty tarvetta toimenpiteille. Suurien objektimäärien käsittelyssä kuitenkin esiintyi viiveitä, ja mahdollisena tulevaisuuden kehityskohteena tunnistettiin suurien objektimäärien käsittelyn optimointi.

6.4 Tutkimusprosessi

Tässä osiossa arvioidaan, kuinka tutkimusprosessissa on otettu huomioon suunnittelutieteellisen tutkimuksen ohjenuorat (ks. taulukko 1).

Ensimmäisen ohjenuoran mukaan suunnittelutieteellisen tutkimuksen tulee tuottaa toteuttamiskelpoinen artefakti. Tutkimuksen tuloksena syntyi luvussa 5 esitelty artefakti, joten ensimmäisen ohjenuoran voidaan katsoa täyttyneen.

Toinen ohjenuora painottaa suunnittelutieteellisen tutkimuksen tavoitteeksi teknologiapainotteiset ratkaisut tärkeisiin ja olennaisiin liiketoimintaongelmiin. Kuten luvussa 4.3.1 todettiin, yrityksellä oli selkeä liiketoiminnallinen tarve, jonka täyttämiseksi tutkimusartefakti toteutettiin. Artefakti on moderni web-sovellus, joten se täyttää myös kriteerin teknologiapainotteisesta ratkaisusta. Edellä luvussa 6.1 arvioitiin jo, kuinka artefakti vastaa taustalla olleeseen liiketoimintaongelmaan. Tutkimus täyttää näin ollen myös toisen ohjenuoran kriteerit.

Artefaktin toteuttamisen taustalla oli ongelma yrityksen portfoliosta puuttuvasta tuotteesta, motivaationa kilpailuedun saaminen ja tarjonnan parantaminen. Lisäksi artefaktille oli asetettu laadullisia vaatimuksia kuten käytettävyys sekä tasalaatuiset ja helposti ylläpidettävät tietovarastoratkaisut. Liiketoiminnan tarpeisiin vastaamisen lisäksi artefaktia on arvioitu edellä teknisesti ja käytettävyyden näkökulmasta. Arvioinnin ulkopuolelle jätetään kuitenkin työn tehokkuus ja laatu, sillä näiden tehokas arviointi vaatisi ensin laajemman projektin toteutuksen. Arviointi on siis kohdistettu alkuperäisiin ongelmiin, motivaatioihin ja vaatimuksiin, joten arviointi kuvastaa tutkimusartefaktin onnistumista. Arvioinnissa on hyödynnetty viitekehystä (taulukko 2), jonka myötä sitä on toteutettu laaja-alaisesti ja eri vaiheissa. Tutkimus vastaa siis myös kolmannen ohjenuoran kriteerejä.

Neljäs ohjenuora käsittelee tutkimuksen kontribuutioita, joiden tulisi olla selkeitä ja verifioitavia. Edeltävissä luvuissa on esitetty tutkimuksen kontribuutioita, jotka pääosin ovat selkeitä. Kontribuutiot ovat kuitenkin yleisesti hankalasti verifioitavissa, sillä niillä on vahva riippuvaisuus yrityksen liiketoimintaan. Lisäksi itse sovellus ei ole julkisesti saatavilla, joten ulkopuolinen taho ei pysty verifioimaan, kuinka se vastaa tutkimusongelmaan. Edellä mainittuja seikkoja ei kuitenkaan välttämättä voida nähdä tutkimusprosessin ongelmana, sillä

vastaava avoimen lähdekoodin toteutus voitaisiin verifioida myös ulkopuolisten toimesta.

Viidennen ohjenuoran mukaan tutkimuksen tulee olla täsmällistä, eli sen tulee nojata täsmällisten metodien käyttöön artefaktin luomisessa ja sen arvioimisessa. Tutkimusprosessiin pyrittiin saamaan täsmällisyyttä esittelemällä sen vaiheet selkeästi prosessin vaiheiden nojautuessa vahvasti tutkimuskirjallisuudessa esitettyyn prosessimalliin. Artefaktin toteutuksessa pyrittiin noudattamaan ohjelmistokehityksen hyviksi havaittuja menetelmiä. Kehitysprosessi oli ketterä ja sisälsi jatkuvaa arviointia. Lisäksi sovellus on suunniteltu noudattaen selkeitä ja hyviksi havaittuja arkkitehtuuriratkaisuja kuten MVC-mallia ja Repository-mallia. Näin ollen myös tämän ohjenuoran kriteerit voidaan katsoa täytetyksi.

Kuudes ohjenuora kuvaa suunnittelutieteellistä tutkimusta etsintäprosessina, jolla pyritään saavuttamaan haluttu lopputulos. Tutkimuksessa etsintäprosessi on selkeästi esillä. Ensinnäkin tutkimus on jaettu kahteen, selkeästi toisistaan erilliseen sykliin. Tämä viittaa jo selkeästi etsintäprosessiin. Lisäksi kunkin syklin aikana toteutettiin useita pienempiä iterointikierroksia, jossa pyrittiin priorisoimaan uusien toimintojen toteutusta ja vanhojen toimintojen parantamista optimaalisen tuloksen saavuttamiseksi. Tutkimuksen aikana on siis pysynyt koko ajan selkeästi esillä viite siihen, että tarkoituksena on toteuttaa optimaalinen ratkaisu.

Seitsemännessä ohjenuorassa todetaan, että suunnittelutieteellinen tutkimus tulee kommunikoida tehokkaasti useille eri yleisöille. Tässä tutkielmassa on pyritty esittämään selkeästi tietovarastoinnin tausta ja määritelmä. Lisäksi on käsitelty siihen liittyviä toimenpiteitä ja arkkitehtuureja sekä kuvattu näihin liittyviä olennaisimpia käsitteitä. Aihe ei ole erityisen tekninen, mutta tekniseen toteutukseen liittyvät yksityiskohdat ja termit on pyritty avaamaan niin, että lukijalla ei tarvitse olla laajaa ennakkotuntemusta tietovarastoinnista tai sovelluskehityksestä. Tutkimus pyrittiin siis kommunikoimaan niin selkeästi, että sen lukeminen on sujuvaa ja helppoa. Seitsemännen ohjenuoran voidaan siis katsoa toteutuneen tutkimuksessa.

Edellä mainittujen seikkojen perusteella kaikki suunnittelutieteellisen tutkimuksen ohjenuorat on otettu huomioon. Tutkimusprosessi myös täyttää kaikkien vaiheiden kriteerit. Tämän perusteella voidaan sanoa tutkimuksen vastaavan suunnittelutieteelliselle tutkimukselle esitettyjä ohjenuoria hyvin.

7 Yhteenveto ja pohdintaa

Tutkielmassa esitettiin suunnittelutieteellinen tutkimus, jonka tuloksena syntyi sovellus tietovarastokehityksen automatisointiin ja metadatan hallintaan. Sovellus kehitettiin yrityksen tarpeisiin liiketoiminnallisten tarpeiden motivoimana. Tutkimuksessa noudatettiin suunnittelutieteellisen tutkimuksen menetelmiä; se kehitettiin kahdessa syklissä ja lopputulosta arvioitiin sovellukselle tunnistettuja motivaatioita ja tavoitteita vasten. Taustatietona tutkielmassa tehtiin kirjallisuuskatsaus tietovarastointiin ja tietovarastokehityksen automatisointiin.

Vaikka kirjallisuudessa onkin käsitelty tietovarastokehityksen automatisointia ja se on todettu tärkeäksi aiheeksi (esimerkiksi Rahman ja Rutz 2015), tutkimusta varsinaisista automatisointisovelluksista on saatavilla heikosti. Suuri osa automatisoinnin tutkimuksesta keskittyy kuvaamaan menetelmiä johonkin yksittäiseen kehitysprosessin toimenpiteeseen, mutta tuotetut menetelmät ovat irrallisia eivätkä sellaisenaan tarjoa hyviä mahdollisuuksia kehityksen automatisointiin kokonaisuutena. Esimerkiksi Phipps ja Davis (2002) sekä Aadil ym. (2016) esittivät yksittäisiä menetelmiä tietomallien luomiseen, kun taas Tomingas, Kliimask ja Tammet (2015) keskittyivät ETL-prosessien SQL-koodin generoimiseen.

Automaatiota käsittelevässä kirjallisuudessa tunnistetaan usein metadatan tärkeys (esimerkiksi Sen ja Sinha 2005; Chaudhuri ja Dayal 1997). Petrović ym. (2017) painottivat, että tehokkaan automaatiotyökalun tulee tarjota muitakin ominaisuuksia kuin koodin generoiminen. Tutkimuksen tärkeimpänä kontribuutiona esitetäänkin kuvaus sovelluksesta, joka koostaa yhteen useita eri toimintoja, joiden avulla automaatiota ja metadataa voidaan hyödyntää laajemmin koko kehitysprosessin ajan. Lisäksi tutkimuksessa tuodaan esille suunnitteluratkaisuja, joiden avulla tietovarastojen kehittäminen pyritään saamaan mahdollisimman tehokkaaksi.

Sovelluksen arvioinnissa havaittiin useita mahdollisia jatkokehityskohteita. Etenkin käytettävyyden testauksessa käyttäjien palautteen perusteella havaittiin käyttöliittymässä alueita, joita uudelleensuunnittelemalla sovelluksen käytettävyyttä voitaisiin edelleen parantaa ja täten sovelluksen käyttöä tehostaa. Kaikkia näitä seikkoja ei kuitenkaan tutkielman rajoissa ehditty toteuttamaan. Myöskään kaikkia kuviossa 2 määritettyjä ominaisuuksia ei vielä ole

toteutettu, joten sovellus ei vielä vastaa kaikkia sille asetettuja toiminnallisia tarpeita. Sovellusta tuleekin vielä tässä vaiheessa käsitellä prototyyppinä, vaikka sitä hyödynnetäänkin jo asiakastyössä ja olennaisimmat metadatan hallintaan ja kehityksen automatisointiin tarvittavat ominaisuudet on toteutettu. Käytettävyyden arviointia varten myöhemmin olisi hyödyllistä toteuttaa betatestauksen lisäksi myös ohjattu käytettävyystesti, jota pidetään kirjallisuudessa (esim. Følstad 2017) käyttäjien omatoimista palautetta tehokkaampana metodina.

Rajoitteena tutkielman tuloksille voidaan pitää sitä, että kehitetyllä sovelluksella ei vielä tutkielman julkaisuvaiheessa ole toteutettu laajaa tietovarastointiprojektia. Näin ollen sen vaikutuksia sosiaalisiin suhteisiin ei ole voitu kunnolla arvioida, sillä nämä vaikutukset ovat selkeästi nähtävillä vasta, kun sovellus on laajemmassa käytössä. Useiden projektien toteutuksen puute rajoittaa jossain määrin myös arviointia liiketoiminnallisesta näkökulmasta. Kuitenkin jo prototyyppivaiheen sovellukselle liiketoiminnan edustajien arviot siitä, kuinka sovellus vastaa liiketoiminnan tarpeita, olivat positiivisia ja sovellus koetaan tärkeäksi osaksi yrityksen tarjontaa.

Selkeä jatkotutkimusmahdollisuus olisikin tapaustutkimus, jossa sovellusta hyödyntäen toteutettaisiin laaja tietovarastoprojekti. Laajan projektin toteutuksen analysoinnilla voitaisiin saada arvokasta tietoa automatisoinnin hyödyistä, etenkin arvioimalla vaikutuksia työn tehokkuuteen ja projektin toteutuksen laatuun. Optimaalitilanteessa voitaisiin siirtyä käyttämään sovellusta ympäristössä, jossa metadatan hallintaan on aiemmin käytetty jotain muuta työkalua tai sellaista ei ole ollut käytettävissä. Laajan projektin yhteydessä on myös mahdollisuus henkilöstövaihdoksiin projektitiimissä, jolloin voitaisiin arvioida metadatan hallinnan ja automatisoinnin hyötyjä projektin kehittäjien vaihtuessa. Tällaisessa tutkimuksessa voitaisiin siis myös perehtyä tarkemmin sovelluksen vaikutuksiin sosiaalisessa ympäristössä.

Lähteet

Aadil, B., A. A. Wakrime, L. Kzaz ja A. Sekkaki. 2016. “Automating Data warehouse design using ontology”. Teoksessa *2016 International Conference on Electrical and Information Technologies (ICEIT),* 42–48. Toukokuu. doi:10.1109/EITech.2016.7519618.

Agarwal, Ritu, ja Vasant Dhar. 2014. “Editorial—Big Data, Data Science, and Analytics: The Opportunity and Challenge for IS Research”. *Information Systems Research* 25 (3): 443–448. doi:10.1287/isre.2014.0546.

Alsqour, M., K. Matouk ja M. L. Owoc. 2012. “A survey of data warehouse architectures — Preliminary results”. Teoksessa *2012 Federated Conference on Computer Science and Information Systems (FedCSIS),* 1121–1126. Syyskuu. https://ieeexplore.ieee.org/abstract/document/6354451.

Ariyachandra, Thilini, ja Hugh Watson. 2010. “Key organizational factors in data warehouse architecture selection”. *Decision Support Systems* 49 (2): 200–212. doi:10.1016/j.dss.2010.02.006.

Baskerville, Richard, Abayomi Baiyere, Shirley Gregor, Alan Hevner ja Matti Rossi. 2018. “Design Science Research Contributions: Finding a Balance between Artifact and Theory”. *Journal of the Association for Information Systems* 19 (5). https://aisel.aisnet.org/jais/vol19/iss5/3/.

Breslin, Mary. 2004. “Data warehousing battle of the giants: Comparing the Basics of the Kimball and Inmon Models”. *Business Intelligence Journal* 9 (1): 6–20. https://www.researchgate.net/publication/237544154_Data_Warehousing_Battle_of_the_Giants_Comparing_the_Basics_of_the_Kimball_and_Inmon_Models.

Bruun, Anders, Peter Gull, Lene Hofmeister ja Jan Stage. 2009. “Let Your Users Do the Testing: A Comparison of Three Remote Asynchronous Usability Testing Methods”. Teoksessa *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems,* 1619–1628. CHI ’09. ACM. doi:10.1145/1518701.1518948.

Castellanos, Malu, Alkis Simitsis, Kevin Wilkinson ja Umeshwar Dayal. 2009. “Automating the Loading of Business Process Data Warehouses”. Teoksessa *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology,* 612–623. EDBT ’09. ACM. doi:10.1145/1516360.1516431.

Chandra, Pravin, ja Manoj K. Gupta. 2018. “Comprehensive survey on data warehousing research”. *International Journal of Information Technology* 10 (2): 217–224. doi:10.1007/s41870-017-0067-y.

Chaudhuri, Surajit, ja Umeshwar Dayal. 1997. “An Overview of Data Warehousing and OLAP Technology”. *SIGMOD Rec.* 26 (1): 65–74. doi:10.1145/248603.248616.

Conboy, Kieran, Rob Gleasure ja Eoin Cullina. 2015. “Agile Design Science Research”. Teoksessa *New Horizons in Design Science: Broadening the Research Agenda,* toimittanut Brian Donnellan, Markus Helfert, Jim Kenneally, Debra VanderMeer, Marcus Rothenberger ja Robert Winter, 168–180. Cham: Springer International Publishing. ISBN: 978-3-319-18714-3.

Følstad, Asbjørn. 2017. “Users’ design feedback in usability evaluation: a literature review”. *Human-centric Computing and Information Sciences* 7 (1). doi:10.1186/s13673-017-0100-y.

Golfarelli, Matteo, Simone Graziani ja Stefano Rizzi. 2016. “Starry Vault: Automating Multidimensional Modeling from Data Vaults”. Teoksessa *Advances in Databases and Information Systems,* toimittanut Jaroslav Pokorný, Mirjana Ivanović, Bernhard Thalheim ja Petr Šaloun, 137–151. Cham: Springer International Publishing. ISBN: 978-3-319-44039-2.

Gregor, Shirley, ja Alan R Hevner. 2013. “Positioning and presenting design science research for maximum impact”. *MIS quarterly* 37 (2): 337–355. doi:10.25300/MISQ/2013/37.2.01.

Hevner, Alan, ja Samir Chatterjee. 2010. “Design Science Research in Information Systems”. Teoksessa *Design Research in Information Systems: Theory and Practice,* 9–22. Springer US. doi:10.1007/978-1-4419-5653-8_2.

Hevner, Alan, Salvatore March, Jinsoo Park ja Sudha Ram. 2004. “Design Science in Information Systems Research”. *MIS Quarterly* 28 (1). https://aisel.aisnet.org/misq/vol28/iss1/6/.

Inmon, William H. 2002. *Building the data warehouse, 3rd edition.* John wiley & sons. ISBN: 0471081302.

Jovanovic, Vladan, ja Ivan Bojicic. 2012. “Conceptual data vault model”. *SAIS 2012 Proceegings* 22. https://aisel.aisnet.org/sais2012/22.

Jukic, Nenad. 2006. “Modeling strategies and alternatives for data warehousing projects”. *Communications of the ACM* 49 (4): 83–88. doi:10.1145/1121949.1121952.

Jörg, Thomas, ja Stefan Dessloch. 2008. “Towards Generating ETL Processes for Incremental Loading”. Teoksessa *Proceedings of the 2008 International Symposium on Database Engineering & Applications,* 101–110. ACM. doi:10.1145/1451940.1451956.

Kakish, Kamal, ja Theresa A Kraft. 2012. “ETL evolution for real-time data warehousing”. Teoksessa *Proceedings of the Conference on Information Systems Applied Research.* http://proc.conisar.org/2012/.

Kemper, A., ja T. Neumann. 2011. “HyPer: A hybrid OLTP OLAP main memory database system based on virtual memory snapshots”. Teoksessa *2011 IEEE 27th International Conference on Data Engineering,* 195–206. Huhtikuu. doi:10.1109/ICDE.2011.5767867.

Kimball, Ralph, ja Margy Ross. 2011. *The data warehouse toolkit: the complete guide to dimensional modeling.* Luku 1. John Wiley & Sons.

Klecun, Ela, ja Tony Cornford. 2005. “A critical approach to evaluation”. *European Journal of Information Systems* 15 (3): 229–243. doi:10.1057/palgrave.ejis.3000540.

Krneta, Dragoljub, Vladan Jovanovic ja Z Marjanovic. 2016. “An Approach to Data Mart Design from a Data Vault”. *INFOTEH-Jahorina* 15. https://infoteh.etf.ues.rs.ba/zbornik/2016/radovi/RSS-1/RSS-1-4.pdf.

Krneta, Dragoljub, Vladan Jovanovic ja Zoran Marjanovic. 2014. “A direct approach to physical Data Vault design.” *Computer Science and Information Systems* 11 (2): 569–599. doi:10.2298/CSIS130523034K.

Kuechler, Bill, ja Vijay Vaishnavi. 2011. “Promoting relevance in IS research: An informing system for design science research”. *Informing science: The international journal of an emerging transdiscipline* 14 (1): 125–138. doi:10.28945/1498.

Larson, Deanne, ja Victor Chang. 2016. “A review and future direction of agile, business intelligence, analytics and data science”. *International Journal of Information Management* 36 (5): 700–710. doi:10.1016/j.ijinfomgt.2016.04.013.

Lee, Allen S., Manoj Thomas ja Richard L. Baskerville. 2015. “Going back to basics in design science: from the information technology artifact to the information systems artifact”. *Information Systems Journal* 25 (1): 5–21. doi:10.1111/isj.12054.

Leoz, Gerard De, ja Stacie Petter. 2018. “Considering the social impacts of artefacts in information systems design science research”. Toimittanut Ken Peffers, Tuure Tuunanen ja Bjoern Niehaves. *European Journal of Information Systems* 27 (2): 154–170. doi:10.1080/0960085X.2018.1445462.

Linstedt, Daniel, ja Michael Olschimke. 2016. *Builiding a scalable data warehouse with Data Vault 2.0.* Morgan Kaufmann. ISBN: 978-0-12-802510-9.

March, Salvatore T., ja Gerald F. Smith. 1995. “Design and natural science research on information technology”. *Decision Support Systems* 15 (4): 251–266. doi:10.1016/0167-9236(94)00041-2.

McAfee, Andrew, Erik Brynjolfsson, Thomas H Davenport, DJ Patil ja Dominic Barton. 2012. “Big data: the management revolution”. *Harvard business review* 90 (10): 60–68. https://hbr.org/2012/10/big-data-the-management-revolution.

Microsoft. 2010. “The Repository Pattern”. Viitattu 19. lokakuuta 2019. https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff649690(v=pandp.10)?redirectedfrom=MSDN.

———. 2016. “Overview of Entity Framework Core”. Viitattu 20. lokakuuta 2019. https://docs.microsoft.com/en-us/ef/core/.

Microsoft. 2018. “API design guidance - Best practices for cloud applications”. Viitattu 19. lokakuuta 2019. https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design.

———. 2019. “Overview of ASP.NET Core MVC”. Viitattu 19. lokakuuta 2019. https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-3.0.

Nielsen, Jakob. 1993. *Usability Engineering*. Luku 5. San Francisco: Morgan Kaufmann. ISBN: 0-12-518406-9.

Palmer, J. 2002. “Designing for Web site usability”. *Computer* 35 (7): 102–103. doi:10.1109/MC.2002.1016906.

Pankov, Ivan, Pavel Saratchev, Filip Filchev ja Paul Fatacean. 2014. “TOWARDS A GENERIC METADATA WAREHOUSE”. *Materials, Methods & Technologies* 8:68–83. https://www.scientific-publications.net/en/article/1000148/.

Peffers, Ken, Tuure Tuunanen, Marcus A. Rothenberger ja Samir Chatterjee. 2007. “A Design Science Research Methodology for Information Systems Research”. *Journal of Management Information Systems* 24 (3): 45–77. doi:10.2753/MIS0742-1222240302.

Peralta, Verónika, Alvaro Illarze ja Raúl Ruggia. 2003. “On the Applicability of Rules to Automate Data Warehouse Logical Design.” Teoksessa *The 15th Conference on Advanced Information Systems Engineering (CAiSE ’03), Workshops Proceedings,Information Systems for a Connected Society*. https://www.researchgate.net/publication/220920349_On_the_Applicability_of_Rules_to_Automate_Data_Warehouse_Logical_Design.

Petrović, Marko, Milica Vučković, Nina Turajlić, Slađan Babarogić, Nenad Aničić ja Zoran Marjanović. 2017. “Automating ETL processes using the domain-specific modeling approach”. *Information Systems and e-Business Management* 15 (2): 425–460. doi:10.1007/s10257-016-0325-8.

Phipps, Cassandra, ja Karen C Davis. 2002. “Automating data warehouse conceptual schema design and evaluation.” Teoksessa *Design and Management of Data Warehouses 2002, Proceedings of the 4th Intl. Workshop DMDW'2002,* 2:23–32. https://www.researchgate.net/publication/220841982_Automating_data_warehouse_conceptual_schema_design_and_evaluation.

Plattner, Hasso. 2009. “A Common Database Approach for OLTP and OLAP Using an In-memory Column Database”. Teoksessa *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data,* 1–2. ACM. doi:10.1145/1559845.1559846.

Pries-Heje, Jan, Richard Baskerville ja John R Venable. 2008. “Strategies for Design Science Research Evaluation.” Teoksessa *ECIS 2008 Proceedings,* 255–266. 87. https://aisel.aisnet.org/ecis2008/87.

Puonti, Mikko, Timo Raitalaakso, Timo Aho ja Tommi Mikkonen. 2016. “Automating Transformations in Data Vault Data Warehouse Loads.” Teoksessa *EJC,* 215–230. doi:10.3233/978-1-61499-720-7-215.

Rahman, Nayem, ja Dale Rutz. 2015. “Building data warehouses using automation”. *International Journal of Intelligent Information Technologies* 11 (2). doi:10.4018/IJIIT.2015040101.

Rifaie, M., K. Kianmehr, R. Alhajj ja M. J. Ridley. 2008. “Data warehouse architecture and design”. Teoksessa *2008 IEEE International Conference on Information Reuse and Integration,* 58–63. Heinäkuu. doi:10.1109/IRI.2008.4583005.

El-Sappagh, Shaker H Ali, Abdeltawab M Ahmed Hendawi ja Ali Hamed El Bastawissy. 2011. “A proposed model for data warehouse ETL processes”. *Journal of King Saud University-Computer and Information Sciences* 23 (2): 91–104. doi:10.1016/j.jksuci.2011.05.005.

Sen, Arun, ja Atish P. Sinha. 2005. “A comparison of data warehousing methodologies”. *Communications of the ACM - The disappearing computer* 48 (3): 79–84. doi:10.1145/1047671.1047673.

Simitsis, A., P. Vassiliadis ja T. Sellis. 2005. “Optimizing ETL processes in data warehouses”. Teoksessa *21st International Conference on Data Engineering (ICDE’05),* 564–575. Huhtikuu. doi:10.1109/ICDE.2005.103.

Simitsis, Alkis. 2003. “Modeling and managing ETL processes.” Teoksessa *Proceedings of the VLDB 2003 PhD Workshop.* http://ceur-ws.org/Vol-76/.

Simsion, Graeme, ja Graham Witt. 2004. *Data modeling essentials.* Luku 1. Elsevier.

Singh, Sandeep. 2011. “Data warehouse and its methods”. *Journal of Global Research in Computer Science* 2 (5): 113–115. https://jgrcs.info/index.php/jgrcs/article/view/97.

Smilowitz, Elissa D., Michael J. Darnell ja Alan E. Benson. 1994. “Are we overlooking some usability testing methods? A comparison of lab, beta, and forum tests”. *Behaviour & Information Technology* 13 (1-2): 183–190. doi:10.1080/01449299408914597.

Sonnenberg, Christian, ja Jan vom Brocke. 2012. “Evaluations in the Science of the Artificial – Reconsidering the Build-Evaluate Pattern in Design Science Research”. Teoksessa *Design Science Research in Information Systems. Advances in Theory and Practice,* toimittanut Ken Peffers, Marcus Rothenberger ja Bill Kuechler, 381–397. doi:10.1007/978-3-642-29863-9_28.

Subotic, Danijela, Vladan Jovanovic ja Patrizia Poscic. 2014. “DATA WAREHOUSE AND MASTER DATA MANAGEMENT EVOLUTION-A META-DATA-VAULT APPROACH.” *Issues in Information Systems* 15 (2). http://iacis.org/iis/2014/81_iis_2014_14-23.pdf.

Tomingas, Kalle, Margus Kliimask ja Tanel Tammet. 2015. “Data integration patterns for data warehouse automation”. Teoksessa *New Trends in Database and Information Systems II,* 41–55. Springer. doi:10.1007/978-3-319-10518-5_4.

Vaisman, Alejandro, ja Esteban Zimányi. 2014. “Introduction”. Teoksessa *Data Warehouse Systems: Design and Implementation,* 3–11. Springer Berlin Heidelberg. ISBN: 978-3-642-54655-6.

Venable, John, Jan Pries-Heje ja Richard Baskerville. 2012. “A Comprehensive Framework for Evaluation in Design Science Research”. Teoksessa *Design Science Research in Information Systems. Advances in Theory and Practice,* toimittanut Ken Peffers, Marcus Rothenberger ja Bill Kuechler, 423–438. Toukokuu. doi:10.1007/978-3-642-29863-9_31.